

Master of Orion 1's game code is a hard thing to write about. It's procedural.

It's all about "then", "for each", "afterwards", "now", "again", not about "why" and "how". Follow the script and you'll get the intended result. Skip a step, you're in an undefined zone.

Humans have much easier time comprehending concepts than following a lengthy cooking recipe.

I try to abstract most important concepts, but if I'll abstract everything, I'd be describing some virtual AI, not the MOO1's one. So, some parts will be more difficult to read than others.

And it's important for me to describe exactly MOO1, because I have things to say and you don't really have a reason to believe me.

MOO1, the game I love deeply and the game you probably at least heard about, gives us a verifiable common ground. You can always check the code and tweak how it works and come to your own conclusions.

I very much wanted to read something like that about 4X AIs and get to my own conclusions. I couldn't find anything like that, so I wrote this essay. Maybe it'll give you a puzzle piece that was missing from your design.

Skip the parts that bore you. They probably aren't that important for you and if you wish, you can always look up what you want here or in the code.

I separated my thoughts into (I think) isolated segments. My concluding thoughts about 4X AIs are at the end of the document, just above the appendix.

They may or may not be aligned with the views of Steve Barcia, Kilgore Trout or people whose comments and advice made this essay readable. However, they're what I think.

Have fun.

Intro	4
My assumptions	4
Who really needs a strong AI?	4
Choosing the tools	6
Sloppy precision	8
Here's the map	10
Phases of the AI	11
Turn-to-turn fleet decisions (p1 phase)	12
Main concepts	13
Fronts	13
Combat value of ships	14
Moving fleets	16

Cadence	16
Scouting	16
Colonization	17
Attacks	19
Wishlisting	19
Main attacks	20
Attacks of opportunity	20
Defense	21
Restocking	22
Invasions and population transfers	23
Housekeeping	24
The bug	24
Thoughts	25
Ship design (p2 phase)	26
Scrapping ships	28
Designing ships	28
Choosing the hull	29
Choosing armor	29
Choosing modules	29
Choosing specials	30
Special detour	32
Weapons selection	32
Bloopers	33
Building stargates and ships	34
Thoughts	34
Battle handling	35
AI vs AI	37
Outcome	37
Counting	37
AI vs human	38
High-level logic	39
Distance calculation	40
Finding a rival	40
Damage giving and dealing	41
Calculating the optimal range	42
Miscellaneous	43
Thoughts	43
“Inspiration” recommendations	45
Colonies spending, research, eco (p3 phase)	45
Planetary spending	46
Research spending	49
Thoughts	50

Spying	50
Spending and priorities	51
Questioning authorities	52
Spying process	52
Converting BCs into spies	52
Infiltration	53
Espionage and tech grouping	53
Sabotage and revolts	54
Thoughts	55
Diplomacy	57
Main concepts	59
Diplomatic act	59
Primary moods of the AI	61
Secondary moods of the AI	62
Diplomatic rapport and traits	63
AI's diplomatic process	63
Deals between the AIs	63
Starting wars between AIs	64
Senate elections voting	65
AI's deals with the human player	66
Starting/ending a war	66
Making a treaty	67
Human player's deals with the AI	68
Mood check	68
Diplomatic options	69
Non-diplomatic options	69
Thoughts	69
Afterword/soapbox	70
Paradigms	71
Optimizations	72
Features	73
Conclusion	73
Appendix	74
Homeworlds assignment	74
Traits assignment	76
Pi equals 3 and other distance shenanigans	78
Futile elegance of base maintenance costs	80
Tech groups and ranking by the AI	81
States of the AI	86
Planetary governor	89
How to familiarize yourself with an existing codebase	91
Build	92

Explore	93
Rename	94
Document	95

Intro

AI is one of the main hurdles for anyone willing to create a new 4X game. It's really hard to figure out how to make the computer follow the gameplay you have in mind. And the AI isn't the only thing in the game, there are a lot of other moving parts. How to decide when to stop?

AI is one of the main complaints for anyone willing to talk about 4X games as well. Lower difficulties are usually perceived as unfulfilling and higher difficulties are usually perceived as computer cheating.

In the vast majority of cases, it does cheat indeed.

We begrudgingly accept the AI in 4X as what it is, we wish we had something better and we rarely get it. Sometimes players and developers even debate what are the goals of the AI at all, stating what they think they want, but, it looks like, not what they really want.

It's fine. It's about time to talk about it.

My assumptions

I'm addressing you as a developer, because, let's face it, who doesn't want to design a 4X game? Maybe just those who are elbow-deep in their 4X game.

I also assume you played Master of Orion 1 or Remnants of the Precursors at least a couple of times, so I don't need to explain how the game works from the player's point of view. Much smarter people than I spent almost 400 pages on that.

Yup, I'm talking about these 400 pages. I heavily recommend the official "Master of Orion Strategy Guide" by Emrich and Hughes for any deeply interested reader. It is a magnificent work written by people who knew this game back when it was called "Star Lord".

I'm not sure whether it is permitted to link to it (the paperback is still listed on Amazon?), but it's quite possible to find it on various archival sites. Kilgore Trout provides a bit of [errata](#) for it.

Who really needs a strong AI?

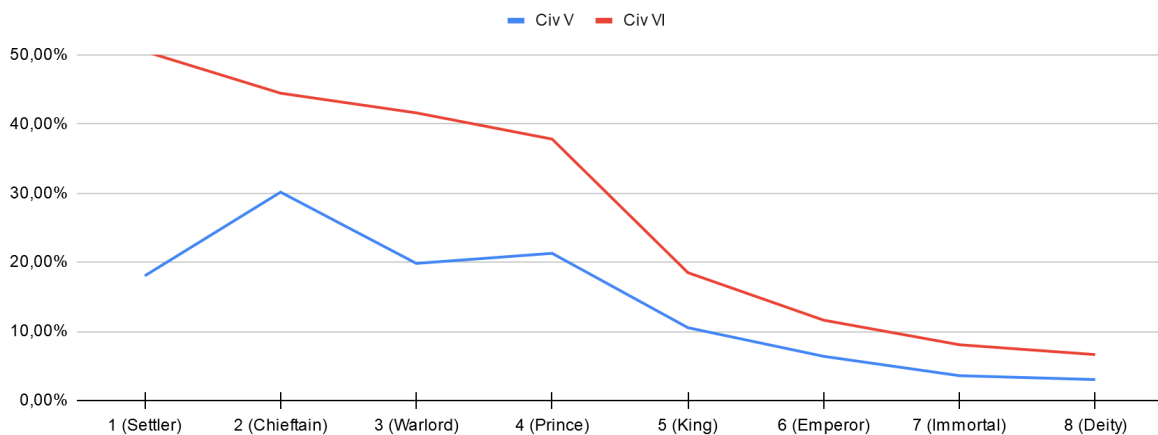
Why do strategy games get shipped with a bad AI? Frankly, paraphrasing and probably mangling what Brad Wardell once said when rating Galactic Civilizations AIs, "AI doesn't sell a game". What sells games is usually graphics, genre and mood. AI is what's helping to keep players in-game.

It's not the only thing though: self-imposed challenges, multiplayer or steady stream of role-playing DLCs keep players in-game as well if not better.

Here's another problem, most players do not want to *git gud* in the game. No, really. I gathered some data from some very popular 4X games which have Steam achievements for winning the game at a certain difficulty level.

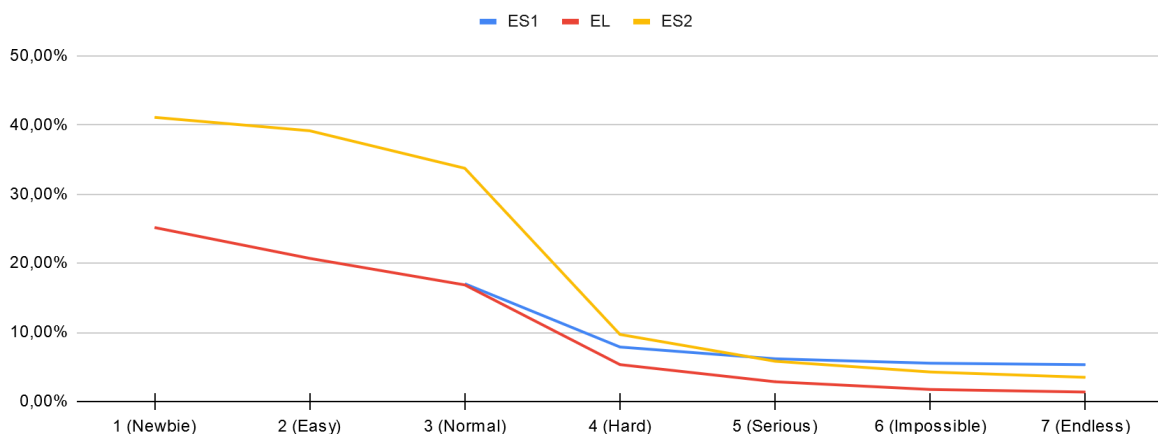
As these games were on sale multiple times and people typically buy stuff at sales to never ever launch the game, I normalized the percent of players having a difficulty level achievement by dividing it by the percent of players having the most popular achievement.

Civilization V and Civilization VI normalized winrate by difficulty level



(Civ V chart is wobbling because getting a higher difficulty win achievement does not grant you the lower ones)

Endless Space 1, Endless Legend and Endless Space 2 normalized winrate by difficulty level



(Endless Space 2 chart's baseline is a bit too low, but with no suitable alternative I went with the first achievement as the baseline as with the other ones; Endless Space 1 does not have 1 and 2 level achievements)

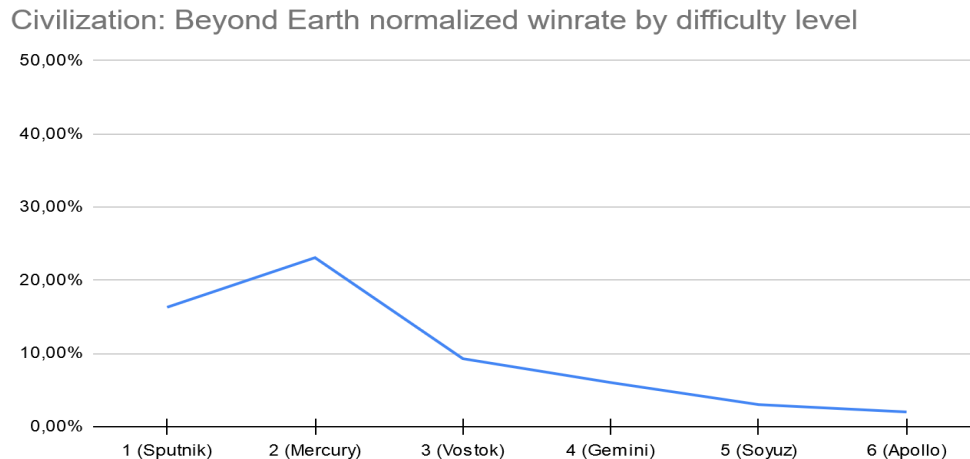
What I see in these graphs are two distinct slightly declining plateaus with a sharp break between the two:

- one from the easiest to normal difficulties,
- one from the hard to hardest difficulties.

For me it indicates that theoretically you can have no effective win condition in a game and the majority of players won't even notice... wait, Stellaris already did that.

Ok, jokes aside, it tells me that just a small minority of players is willing to go higher than “normal” difficulty. And it’s not about the actual challenge to the player, just the word “hard” seems to discourage more than half of the engaged players.

Case in point, Civ: BE’s graph, normalized the same way and scaled like the previous graphs:



No “normal” difficulty, no problem. Players just choose the “not the easiest” one.

So, from the developer’s standpoint making AI more capable as a player has quickly diminishing returns after the level that’s usually considered “normal”. Generally speaking, the AI is meant to lose to the player gracefully, that’s all.

And if you make the AI strong, you must also make more or less transparent why it’s winning, what did it do better than the player. Otherwise your average player will quickly accuse a perfectly honest AI for cheating (like it happened, for example, with Ail’s improvements to Pandora) and, given the genre’s track record, that would be pretty understandable.

The AI’s strength is not what directly sells or supports the game, it’s something you can always hope to do later or even offload on willing mod makers. Better and more optimized particle effects will likely move the needle more than teaching an AI how to trade techs efficiently. The market is arguably unjust, but at least it pays in money and not in exposure. An argument can be made that for promoting the game through youtube or streaming personalities a hard mode is a must, but the math here is even more murky. Talk to your marketing department (if any) or personalities you know (if any).

Making an AI better is what you toil at primarily for yourself and for bragging rights, purely out of your arrogance to make a 4X and out of your pride in the game.

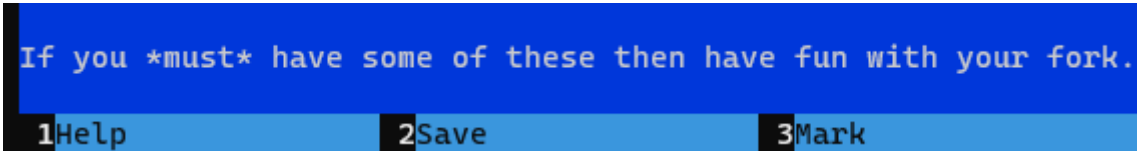
Choosing the tools

Now, you still need an AI. And most probably you really want to implement it the best way you can, but you don’t know exactly how. You’ve got many ideas, but there’s not much to cross-reference them with. If only there was some proven minimal viable AI... wait, there is!

There are even several. Codebases for [Remnants of the Precursors](#) (Java) and [Star Ruler 2](#) (C++) are publicly available and everyone is welcome to tinker with them. But they are made with about twenty years worth of hindsight and are not exactly minimal.

Instead we'll go to the game that coined the genre, the game that's still played and praised for its "endless variation", the one and only Master of Orion 1.

Technically, we're not looking at Master of Orion 1's code. We're looking at an utterly faithful and pedantic recreation of it made by the great and powerful Kilgore Trout, at [1oom](#). For our intents and purposes it's even better.



If you **must** have some of these then have fun with your fork.

I promised that I won't go into the actual code and indeed, I'll stay on the logic level. However, 1oom code is well worth studying. It both reflects the algorithmic approaches of Master of Orion 1's era and exemplary (except for variables' names which are an... acquired taste) coding practices of Kilgore Trout.

I recommend the '[Modern C](#)' book by Jens Gustedt (Creative Commons BY-NC-ND 4.0 licensed) for those who want to follow at home. 1oom build process for Linux is straightforward and [well-documented](#), you just need the original .LBX files from MOO1 to run the game.

If you want to run it on Windows, well, native Windows builds are trickier. I had success with mingw binaries using MSYS2, but it's not as simple. Running an X server on Windows and building and running the game in WSL1 (no audio) or WSL2 (you can use pulseaudio for Windows) is easier.

However, if you're looking for a language to do your 4X game in, don't use C unless you really know and love things like `char const * const * const`.

If you want to make a game, C# most probably will be the optimal choice at the moment, all trade-offs (speed, memory handling, tooling, libraries, support available) considered. Also, multithreading in it is relatively easy. If you don't embrace parallelization early on, chances are later you'll lament you didn't.

Additionally, you have an option of switching to (or switching from, or working with) Unity middleware. DOTS in Unity is not exactly ready for prime-time at the time of writing, but it may be there in a year or two.

One more thing before we start. You can write in whatever language you want, do any kind of AI that you may want, you're completely free. But you absolutely must optimize your damn pathfinding logic.

Because it's going to be called at least several thousand times during a turn.

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
25.00	0.07	0.07	573	0.12	0.20	ui_delay_us_or_click
21.43	0.13	0.06	733	0.08	0.08	video_render_gl_32bpp
14.29	0.17	0.04	72	0.56	0.56	lbxpal_build_colortable
10.71	0.20	0.03	17155492	0.00	0.00	hw_get_time_us
10.71	0.23	0.03	39400	0.00	0.00	lbxgfx_draw_pixels_fmt0
7.14	0.25	0.02	1728	0.01	0.01	lbxgfx_draw_pixels_offs_fmt0
3.57	0.26	0.01	208890	0.00	0.00	util_math_go_line_dist
3.57	0.27	0.01	1871	0.01	0.01	game_design_update_engines
3.57	0.28	0.01	10	1.00	1.00	game_battle_handle_all
0.00	0.28	0.00	208882	0.00	0.00	calc_angle_do
0.00	0.28	0.00	208882	0.00	0.00	util_math_angle_dist_cos
0.00	0.28	0.00	208882	0.00	0.00	util_math_angle_dist_sin
0.00	0.28	0.00	208882	0.00	0.00	util_math_calc_angle
0.00	0.28	0.00	145143	0.00	0.00	hw_video_get_buf
0.00	0.28	0.00	125237	0.00	0.00	lbxfont_plotchar
0.00	0.28	0.00	125237	0.00	0.00	lbxfont_plotchar_1x
0.00	0.28	0.00	125237	0.00	0.00	lbxfont_print_char_ret_x
0.00	0.28	0.00	110589	0.00	0.00	hw_video_set_palette_byte
0.00	0.28	0.00	67780	0.00	0.00	util_math_dist_fast
0.00	0.28	0.00	39400	0.00	0.00	lbxgfx_draw_frame_do

(this is the gprof output of running 100m SDL1 build on a 108 stars galaxy with 5 opponents for 10 turns doing nothing, witness these distance calculation calls)

Even on the simplest terrain “pathfinding” (represented here as checks for distance on a flat plane) will be a significant chunk of what the CPU is going to do.

If you plan to have orbital mechanics or space highways or impenetrable mountain ridges or treacherous swamps or Sargasso of the void or anything fun on the map at all, pretty pretty please, read what you can find about pathfinding algorithms (there’s a lot of relevant literature) and plan accordingly.

Go ham on everything else.

As a side note, the pathfinding routine performance was probably one of the main reasons, if not the main one, for Stellaris to ditch different FTL modes. Starlanes and systems are essentially vertices and nodes of a graph and there are a lot of smart and capable algorithms to deal with graphs.

Also, Stellaris probably cuts some corners when creating the map by generating the most efficient graphs to deal with. This is corroborated by the significant tick time increase when you build a lot of stargates: they introduce new vertices into the graph and it becomes much less elegant to traverse.

But I digress.

Sloppy precision

MOO1 was written well before the game industry at large adopted object-oriented programming. This particular shift arguably happened in the late 90s and for a short while there was horrendous C++ code written in C style.

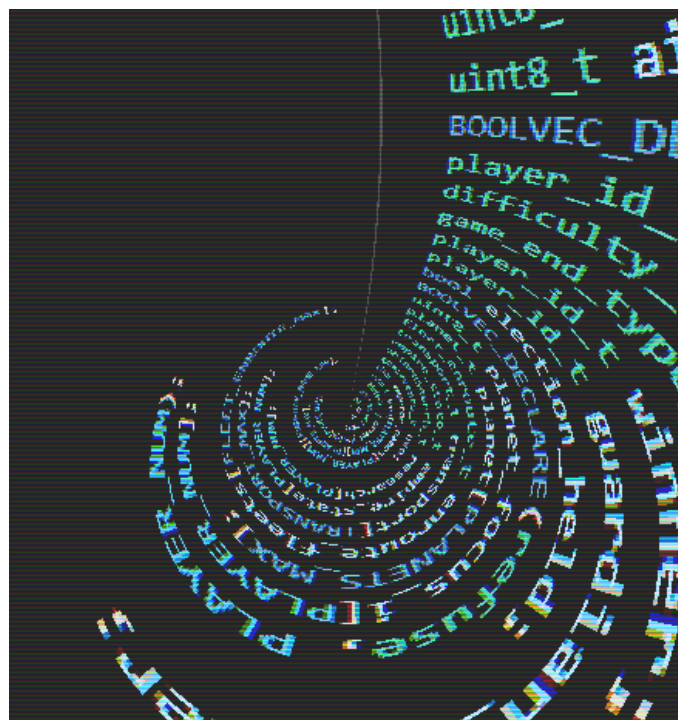
Gamedev tends to be quite conservative in programming practices compared to the programming field overall, because gamedev produces not tools and not services, but entertainment artifacts. Different priorities.

Typically, from the modern programming perspective, you expect the game to be a bunch of well-defined objects politely asking each other “may I?” through compartmentalized methods mostly inherited from parent classes. Eh, if you dabbled in Unity or Unreal Engine, you most likely saw it.

MOO1 in a brutalistic fashion shoves the whole game state (represented as a structure of structure of structures) between functions which may do with it whatever they like. If you had any programming classes, you probably were warned against using global variables whenever possible because it's hard to track what changed what where. Here the whole game is treated kinda like this.

It's a giant Rube Goldberg machine that operates with a sloppy precision. The results of the calculation may be not quite correct, who cares, but the value that'll be written into the game's state is made to be valid at any cost.

This essay is written with sloppy precision too. If you think you learned something here, don't forget to double-check that.



There are other marks of the time in the code (and that's why I'm going to describe what parts of the AI do, not how exactly they do it).

We're used to managed memory nowadays. C#, Java, Python, rewrite-it-in-Rust, JavaScript all automatically allocate and free memory you need for this or that class or structure. Only C++ is by default unmanaged, but in gamedev it's used primarily server-side in MMOs or in Unreal Engine projects, and UE tries to paper over it with its memory management macros and a custom garbage collection.

It's because memory leaks scarred a generation and a half of programmers and they vowed to never deal with it again. Even C has got some unofficial garbage collection solutions since.

But MOO1 is 1993. So it was made to have a largely predetermined and largely constant memory footprint for a given map size/number of players combination. The minimum requirements for the game were 2MB of RAM, not much memory to waste.

Of course, such an extreme decision caused a lot of trade-offs. You can't have more than 260 traveling fleets in game total, more than 100 transport fleets en-route total, etc. What's more important is that the CPU is forced to calculate everything that could be derived from the game state over and over and over again.

For example, an empire doesn't have a list of planets it owns, instead each and every star on the map is queried "is empire_name your owner?" Homeworlds of all things are selected by randomly picking stars, checking if they fulfill the criteria (if after 200 tries no good combination is found, the map is generated anew). Calculating visibility... oh, that's a doozy.

One more thing, there's no floating point arithmetics in the game. Back in the day it was the domain of expensive "mathematical co-processors". An FPU was integrated only in i486 (DX or SL, not the cheaper SX).

Even though it was a bit long in the tooth by 1993 (that year Intel had already started selling i586 with a superscalar architecture better known as Pentiums), it still cost an arm and a leg. A system with an i486 would set you back approximately 5000-6000 in today's USD.

MOO1 could expect its audience to have an FPU just about as much as you can expect your audience to have 4 high-end video cards in the system, so the code is all peppered over by the usage of strictly integer arithmetics.

Any division you see mentioned is an integer division, every numerical value is an integer as well. Where you could see decimals, like in espionage spending, the actual underlying number is an integer number of tenths.

Fascinating and horrifying if you're into this stuff.

```
/* spawn a new colony ship by magic */
research_state->shipcount[colonyship_design_id] = ++colony_ships_num;
++empire_state->orbit[planet_id].ships[colonyship_design_id];
return true;
```

Here's the map

If you really want to look into the code, the start of the game proper is at *main_do()* function in *src/game.c*.

Without the UI-related code and loading from LBX files details, MOO1 works as follows:

- 1) prepare new game (*game_new()* in *src/game/game_new.c*)
- 2) start the game (*game_start()* in *src/game/game.c*)
- 3) run the turn (*game_turn_process()* in *src/game/game_turn.c*)
- 4) if the end of the game isn't reached (*game_end.type* is *NONE*), go to step 3
- 5) if the end of the game is reached, show an ending depending on the end type
- 6) stop the game (*game_stop()* in *src/game/game.c*, but not exactly)

I also prepared a decent map for you. Here's the [full-res vector](#) image and the [full-res raster](#) image.

It's not fully representative of what goes in the code, it doesn't follow any standard of code charts I know about, it's meant to give you an understanding of what's where, but not much more.

I omit calls to the pseudorandom generator (it's well worth a look if you're interested, but irrelevant to the AI), I omit most math calls (unless I want to note some things, like with *util_math_dist_fast()*), I omit UI calls.

Purple arrows pointing downwards mean sequential execution, but in some cases I repeat the function calls if I feel it's necessary and in others I just name what's called without reflecting the underlying logic of the code. Black lines pointing right usually mean a function call. Tacky parallelograms are what I use for sections of the graph that are too repetitive for my liking, but not always.

A bit like MOO1, it's also made with sloppy precision.

Frankly I made it just to find my way back out of these function calls. Maybe it'll be of some use for you too.

Well, I also intended to go function by function when I started writing. Then I found out that even for an expert in this field reading it was "extremely fatiguing". Also I found out that I don't want to read what I wrote, and that's a fairly bad sign.

Instead I'll just stick to the logic. I put things I really didn't have the heart to delete into the appendix.

Phases of the AI

Despite me calling its AI “minimal”, Master of Orion 1 is not a toy project. It’s a fully fledged commercial game that captured the imagination of hundreds of thousands of players, if not millions.

The AI in MOO1, just like in any 4X game, must colonize and develop planets, build and move fleets, declare war and peace, research, spy, hold audiences and more. The beauty of MOO1 is that all these moving parts are separate from each other and are rather simple to comprehend.

It’s better to even call them phases, as, like phases in a boardgame, they happen one after another and do not mix. Most of the AI-related functions are not even called for any player in particular, but cycle through all players in the game and occasionally check things like “IS_HUMAN” or “IS_ALIVE”.

Omitting a lot of housekeeping logic and things you are going to implement in your own style anyway, like fuel or visibility checks, MOO1’s AI can be neatly divided into following phases:

- p1 — turn-to-turn fleet movement: scouting, colonizing, attacking, etc
- p2 — creating new ship designs and scrapping old ship designs
- p3 — colony development guidelines, research priorities, espionage spending
- handling battles — with separate sets of rules for AI vs human and AI vs AI
- espionage — with some diplo routines calling
- diplomacy — with elections and audiences

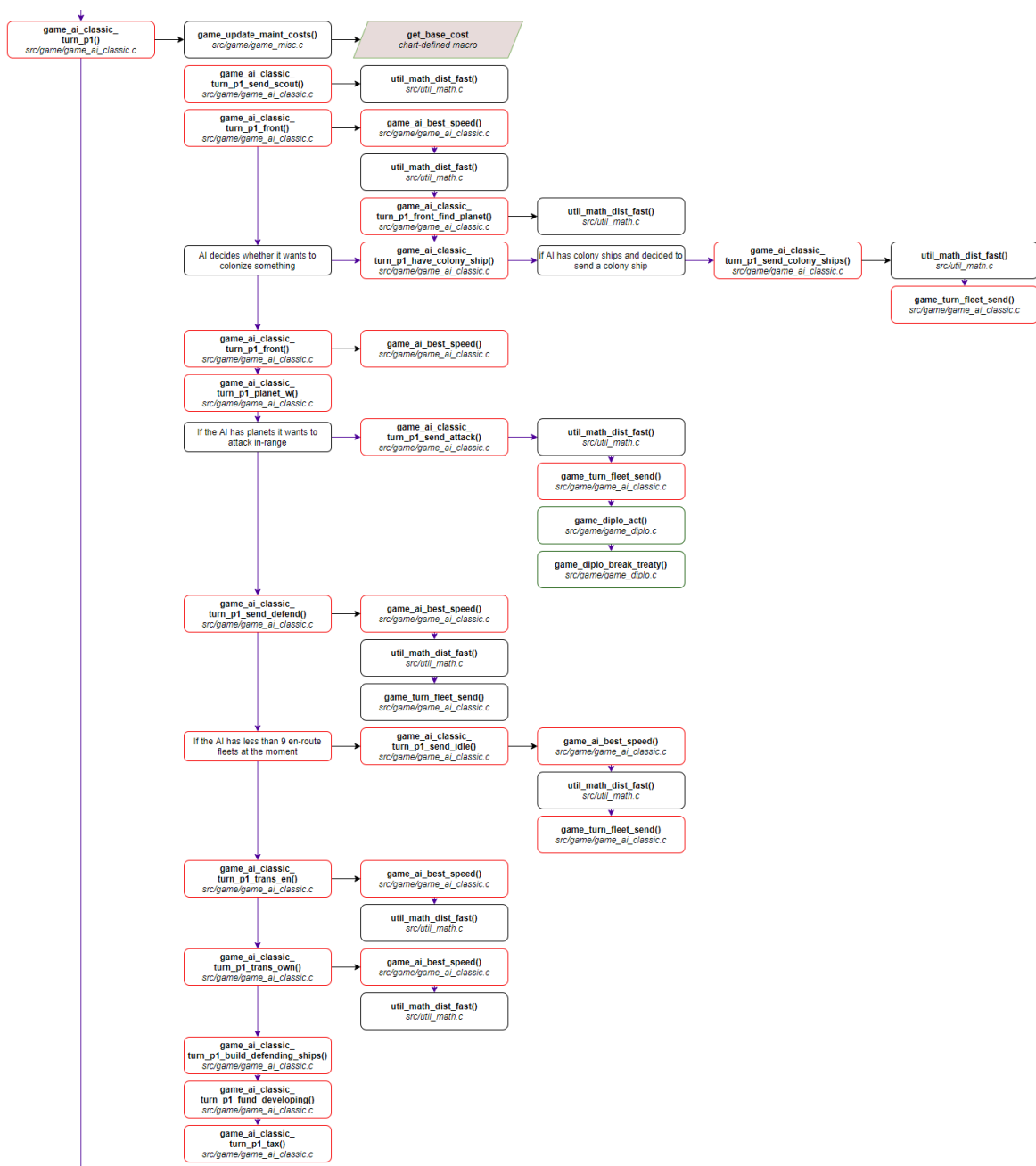
A very good lesson can be learnt from each one.

Technically the planetary governor can be also described as an AI, but it’s set only from UI by the human player. So it’s both part of the game’s AI, but can’t be described as AI. Also, it’s apocryphal.

I’ll just put it into the appendix.

Turn-to-turn fleet decisions (p1 phase)

Here’s how it looks ([full res](#)):



As you can see, the cadence goes as follows:

- 1) scout
- 2) define fronts and front planets and calculate forces at hand
- 3) send colonization ships
- 4) define fronts and front planets and calculate forces at hand again (!)
- 5) attack
- 6) defend
- 7) restock fleets
- 8) send transports to invade enemy colonies
- 9) send transports to bolster own colonies
- 10) do some housekeeping (no, ships aren't built at *build_defending_ships()*)

Coupled with diplomatic audiences, it covers the majority of an AI opponent's visible behavior in the game. How and why does it do things?

We'll start from the concepts behind the decision-making and then the most interested readers can go through the cadence.

The order in the cadence is very important, as at each step the AI deals with what was left or set up by the previous steps.

Main concepts

Fronts

Any AI player in MOO1 (and in 4X overall) has the same goals as the human player. This goal is to paint the map into your color by any means necessary, usually through combat.

The AI treats every empire it has contact with as a potential enemy, potential front that may be opened sooner or later. Front in the military sense, as in the theater of war.

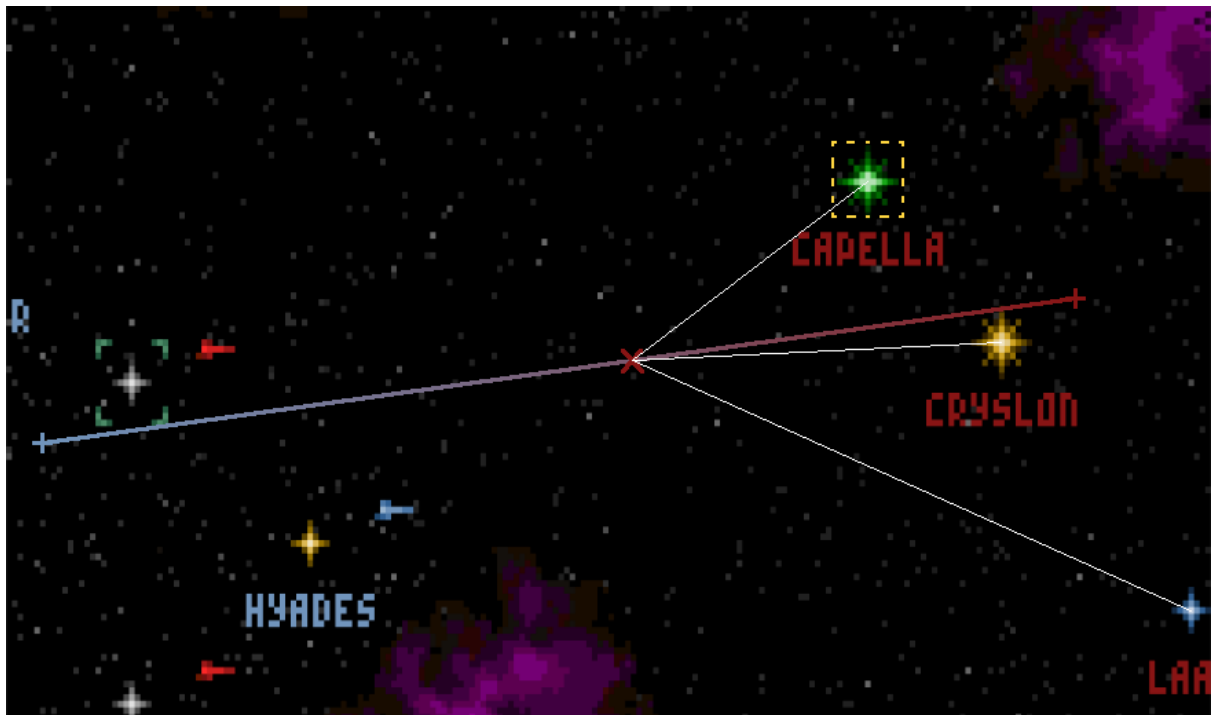
Each front has a designated colony where a fleet will be staged for attack or for defense (mainly for attack, of course). I'll call it the front planet, because that's how it's called in the code.

It makes sense for the front planet to be in the general direction of the enemy and so it is.

The AI calculates the center of its empire by averaging coordinates of all colonies it owns. If the empire is crescent-shaped, the center may very well be even out of reach for the AI. It's fine.

In the same fashion the AI calculates the center of the neighbor and "draws" a line between these two points. Then this line is divided 60:40 with 60 being to the neighbor's center and 40 to this empire's center.

The colony of the AI that's closest to that dividing point is designated the "front planet". This process is repeated every turn as stars tend to change hands and tentacles all the time. Such is the nature of the game.



Now, I bent the truth a little.

MOO1 indeed chooses the closest planet to the front point, but then conducts another search. This time it searches for colonies that have only missile bases, but not a single ship in orbit or for colonies that have no missile bases and just a single ship in orbit. If the AI has such colonies, it will designate the closest one as the front planet for this turn, no matter where it actually is.

I don't really understand which purpose it serves.

The AI half-heartedly tries to balance the forces on front planets. It tries to restock most "weak" front planets first. The relative weakness of the planet is influenced in part by the relations with the target empire. But it doesn't always happen. It's better explained in the "restocking" part.

Combat value of ships

MOO1 has three ways to estimate the combat value of ships: by hull weight, by threat level and the one that's used only in AI vs AI battles.

Hull weight heavily underestimates the ship's capabilities. Threat level underestimates the ship's capabilities a bit less. So, starting from the very early game the threat level outpaces the hull weight.

Only one proper ship design has lesser threat level than hull weight and it's the starting colony ship (24 threat level vs 25 hull weight).

There are several cases when the decision is made by comparing the summary threat level of one side to the hull weight of the other side, and I believe it's deliberate (and later I'll argue it once again to drive the point home).

As you'll see, each AI knows everything on the map all the time. If the AI wasn't made to underestimate opposition and overestimate its own forces, the game would've been very turtly, calculated and dull.

Hull weight is easy.

A small ship has combat value of 1, a medium ship is 5, a large one is 25 and a huge ship is 125.

Threat level is more complex.

It accounts for weapons, shield level, hull hp and specials, but ignores speed, beam attack, beam defense and missile defense values.

More accurately, it's the sum of:

- weapon tech level * number of weapons of such type (bombs don't count)
- $((\text{shield tech level} + 10) * \text{ship HP}) / 50$
- tech level of each special (multiplied by 2 starting from the 9th level)

Note:

- shield class IX has 8th level, class XI has 9th level, class XIII has 10th level and so on
- special tech level 1 is reserve fuel tanks, levels 2-8 are colony bases, level 9 is battle scanner



(This early-game medium ship already has threat value 7: three from lasers plus four from shield and hull)

You probably noticed that the game does not account for the battle computer in any of these two combat ratings. It does in AI vs AI weight calculation.

However, it doesn't account for maneuvering there (or anywhere in the code) for reasons we'll get to in the battle AI overview.

Moving fleets

There are a couple of cases when an AI moves a part of the fleet.

It happens with scouting: the AI selects a stack that can reach the target and sends $\frac{1}{4}$ of it plus 1 ship (so that a single ship gets sent).

It happens with colonization: just the colony ship is sent to the target.

However, most of the time MOO1's AI sends the whole fleet at once (*game_turn_fleet_send()* if you're interested). Whatever was orbiting star A gets an order to go to star B.

What if due to some oversight star B was unreachable for some stacks in the fleet? Before Kilgore Trout's fixes they just vanished into the void.

I like to imagine they just said "screw it" and raised Jolly Roger on their ships.

This produces the constant need to shuffle fleets around.

A front planet may have a grand armada in orbit, but once it raises anchor, this colony becomes very vulnerable and a defense fleet has to be sent in.

First of all, this makes the order of the cadence very important (is it the third time I'm repeating that?), because each time the AI looks at the strategic situation without much prior context; it won't recall an attack if there's not enough ships to defend.

Second, this makes the AI vulnerable to various tactical moves that a human player may use.

Third, the empire looks alive with all these ships scooting around. And that's a really nice thing to have.

However, the AI doesn't really care about the ETA of these fleets. It doesn't coordinate them or anything. There are some checks in place, but they are not exactly measuring turns and contain a bug.

The bug.

Such a bug that it requires its own story.

But we will get to it later.

Cadence

Scouting

The AI scouts only for show.

The main reason to scout the map is to know where the good planets are and AI already knows all there's to be known about the map (including fleets and transports).

So, the only reason why AI does that is to maintain an illusion of an equally capable opponent for the human player.

To get a list of stars it's interested in scouting this turn, the AI goes over all stars on the map and ignores:

- already explored stars
- stars outside of extended fuel range
- Orion before the 150th turn
- stars which are orbited by non-allied fleets
- stars to which this AI's fleets are already flying
- and Orion again, this time unconditionally

Well, now it has a list of target stars and the iteration process repeats. For each target the AI looks for the closest fleet it owns (technically, for the closest star).

Within that fleet, the AI looks for stacks with reserve fuel tanks and selects the last one.

If there's no such ship design in the fleet and the target star is reachable with normal range, it selects just the last stack.

If no ship in this fleet can reach the target star, nothing is sent. The AI won't look any further.

If the AI found a suitable ship stack, it sends a quarter of it plus one ship.

That "plus one" is important. In the early game the AI scouts with the same fleet as you do, and 3 divided by 4 results in 0 with integer division.

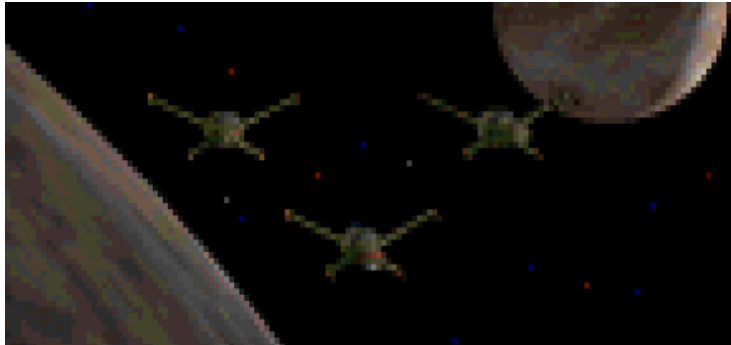
The AI doesn't have a dedicated scout ship design at all. Later in the game the only ships with reserve fuel tanks it'll have will be colony ships.

But as the AI's flight range increases, normal combat fleets will increasingly be used for scouting and that's why just sending off a single ship won't do either.

And it's much better to scout with eleven medium fighters than with just one small scout. You're treated with more respect.

The AI doesn't really fret much about covering all stars in-range in the fastest way. There are dead ends in this system and stars that would've been perfectly scoutable for a human will stay unexplored.

But it doesn't really matter as AI knows what's there anyway.



Colonization

To send a colony ship, the AI has to have three things:

- the mood for colonization
- a planet to colonize
- a colony ship

An AI with the expansionist trait is always in the mood for colonization.
All others inspect their existing colonies first.

The AI categorizes a colony as “developing” if it has less than $\frac{2}{3}$ of the maximum population and less than one missile base per 20 population.

If the AI has more developed colonies than developing ones, it'll go for colonization.
As we'll see later, the AI actively uses population transfer, so it tends to have colonies developed fairly quickly.

Now, the AI needs a planet to colonize and, frankly, it doesn't even bother to check if it was scouted or not. It just knows what planets in-range are good and goes straight to them.

And the last thing the AI needs is a colony ship. It's a big investment for a human player in the early game and even mid game.

The AI isn't smart enough to build enough colony ships in time for the early-game rush, so it cheats. But not always.

First of all, it doesn't go cheating if it has more than 3 colony ships.

Second, if the game difficulty is “Simple” or “Easy” the AI will be stopped from cheating altogether with a 5/6 or 4/6 chance respectively.

Third, if the empire doesn't see colonizable planets nearby, it won't cheat because it doesn't need to.

And lastly it won't cheat if the random number from 1 to 500 is lesser or equal to the $\frac{2}{5}$ of the total production capacity of the empire as it already has enough production capacity.

But if all these checks passed, a new colony ship is poofed into existence by magic.

I already used the picture with that part of code, so instead I'll let the picture tell about the last bit of cheating.

The AI's colony ship always has the best colony base and a single laser. Starting from construction tech level 6 they also have reserve fuel tanks.



SPECIAL NAME	DESCRIPTION	COST	SIZE	POWER	SPACE
none		0	0	0	0
RESERVE FUEL TANKS	EXTENDS SHIP RANGE BY 3 PARSECS	35	435	0	435
STANDARD COLONY BASE	ALLOWS NORMAL PLANET LANDINGS	304	658		658

TOTAL SPACE 1120
AVAILABLE SPACE 382
BUILD

(Spoiler alert: they don't fit at construction tech level 6 even without the laser)

The fronts are recalculated after sending the colony ships because a colony ship has threat value at 24 at least. That's quite a significant chunk of "combat" power in the beginning of the game.

By the way, this subroutine was a source of a well-known bug.

If an AI had more than 3 colony ships, not only it wouldn't cheat, it wouldn't send them to colonize either.

The only way it could get out of this state was if one of the colony ships got destroyed by someone else.

Kilgore Trout's "Classic+" AI fixes this bug.

Attacks

Wishlisting

The empire eyes the galaxy with hungry eyes and adds all stars that are colonizable for it, in-range and owned by a rival or an enemy into the wishlist.

If there's no colony yet, the star is treated as owned by whoever has a fleet orbiting it.

To create a wishlist an empire doesn't have to be in a conquering mood.

However, a wishlist of a non-conquering empire will contain only non-colonized planets.

To want other empires' colonies, the AI must have no planets left to colonize (whether it wants to expand or not) and then with a $\frac{1}{8}$ to $\frac{1}{4}$ chance (from the easiest to the hardest difficulty) it will start making plans.

Also it may already be at regular or final war, that works too.

It's interesting to note that the AI can never attack the Guardian as it is not a player. The AI can attack only the other players who colonized Orion.

Orion will be ignored up until 121st turn.

Now the AI sorts the wishlist by the colony value.

Each population is worth 1 point.
 Each missile base is worth minus 10 points.
 The characteristics of the planet add (*planet_special* * 30 - 20) points.

That's quite useful to know to make your planets undesirable for the enemy.
 Let's put it into a table and compare the results with the recommendations from the official strategy guide:

	Planet special	Special value	Official recommendation	Value for enemy at:		
				50 pops	100 pops	150 pops
Ultra poor	0	-20	1 base per 20 million	0	30	50
Poor	1	10	1 base per 15 million	30	40	60
Abundant	2	40	1 base per 10 million	40	40	40
Artifacts	3	70	1.5 base per 10 million	40	20	-10
Rich	4	100	1.5 base per 10 million	70	50	20
Ultra rich	5	130	2 bases per 10 million	80	30	-20
Orion	6	160	2 bases per 10 million	110	60	10

Not surprisingly, the advice in the guide is solid.
 Not only does it regulate the perceived value of your colonies very well, but it also has a side effect of making your undesirable planets more desirable to the enemy than your powerhouses.

Main attacks

Main attacks are the “proper” ones.

They start with the wishlist we just looked at, are launched from the front planet, are combined with diplomatic declarations and actually aim to kill.

The process of looking for the target for the main attack is a bit strange.

If there's just one planet of this particular rival on the wishlist and it's close enough to the relevant front planet, it is selected as the target.

If there's more than one planet of this rival on the wishlist every single one of them may be ignored with 60% probability, even if it's close enough.

This means if you expose two planets, you have a 36% chance to get away this turn, if three, you have 21.6% and so on. But if you expose just one, well, it will get selected as the attack target all the time.

“Close enough” means roughly 1.5 standard fuel range in parsecs.

To see why “roughly”, look into the distance shenanigans part in the appendix.

For the selected target the AI sums up its defenses (even if there's no colony on the planet):

- sum of threat levels of all ships orbiting that star (excluding owned by this AI)

- $(5 * \text{the target empire's weapon tech level} + 10 * \text{colony shield level} + 100) * \text{missile bases number}$

The AI sums up the threat level of the fleet it has at the front planet as well. If it's higher than the defense level we just calculated, game's on.

Now, if the war is already declared, the AI sends the fleet.

If there's no colony at that star, the AI sends the fleet as well. Non-colonized stars are probably the only valid target for an empire without conquering mood to attack. Remember, the non-aggression pact works only at owned colonies.

But if there is a colony and there is no war yet?

Well, time for diplomacy checks.

If there is an alliance, it gets broken with a 25% chance. If this AI has an expansionist trait, scratch that, it'll break it with a 50% chance. War will be declared and the relations will be reset to 30, if they were higher.

If there is a non-aggression pact, it usually gets broken with a 50% chance and always gets broken by an expansionist. War will be declared and the relations will be reset to 20, if they were higher.

Attacks of opportunity

Attacks of opportunity at the first glance are messy.

If there is a wishlisted planet not yet attacked this turn and the AI has a colony in range of it, with a 60% probability the AI will consider to launch the attack.

Sounds fine so far, but the checks are minimal.

The AI doesn't care if there is a war or treaty or something else and woefully underestimates the enemy.

The defense force will be estimated using hull weight and the attack will be sent if the threat level of the fleet on hand is more than just 2/3 of that already undercounted defense. Missile bases are not even in the equation.

```
for (int k = 0; k < another_empire_state->shipdesigns_num; ++k) {
    defense_force += another_empire_state->orbit[target_planet_id].ships[k] * game_num_tbl_hull_w[enemy_ship_design[k].hull];
    /* FIXME? why not tbl_shipthreat? */
}
```

(Kilgore Trout is puzzled too)

I spent quite a lot of time to find a plausible explanation and I think I found the correct one.

These attacks of opportunity are lowballing the expected opposition level intentionally, because they aren't meant to succeed.

They are meant to keep you, the player, on your toes.

If you have a decent standing fleet or good defenses on a colony you'll feel good when you repel such an attack.

You may or may not be losing strategically, but you'll feel good nonetheless.

If you don't, you'll suffer, you learn not to keep all your eggs in one basket and then you'll feel good when your prudence is validated.

And even if you're not at war and have a non-aggression pact or what have you, you'll get spooked by a sizable fleet of another empire just loitering around your new colony.

Even if they eventually fly away, you kinda get the message that your only true friends in this galaxy are your fleet and your missile bases. Better build them up.

The player must be granted small victories even and especially against a strong AI. Winning small is the only way to start to believe you can win big.

As a side note, it is possible to build an engaging fictional narrative about tactical victories amidst a strategic defeat. Descent: Freespace 1 and 2 are prime examples of this approach.

Defense

The fleets were sent to attack, now it's time to defend.

For each of its colonies, the AI tallies up all military and transport fleets headed to it. Remember, fog of war for the AI is a mere suggestion.

Military fleets are weighted using hull weight, transport ships weigh $5 * \text{number of invaders}$.

Even if no one threatens a colony, if it has less than 5 missile bases and has no orbiting fleet, it gets assigned an incoming threat of 50.

Theoretically that should mean "if you have no fires to put off right now, guard your new colonies".

These colonies rarely get protected though.

After the incoming threat is assessed, the AI looks through all its colonies and if the threat level of their fleets is less than $\frac{3}{4}$ of the threat level, it adds these colonies to the defense list.

Note, the incoming threat is lowballed twice, again!

First by weighting using another measure, then by using $\frac{3}{4}$ multiplier.

The AI seems to be intentionally made squishy on the defense.

Now, for each planet in the defense list a suitable defense fleet is searched in a certain radius. It must come from a non-threatened planet and its threat level should be higher than 160% of the incoming threat.

The AI can't combine fleets, so finding any defenders is not a given.

Threatened planets behave like choosing beggars: "No, this fleet is too small, we don't want it."

The radius is meant to ensure that defense fleets aren't sent with 15 turns of ETA, but it's not that simple.

More on that later.

Restocking

Ok, scouts are flying, colony ships are flying, attack fleets are attacking, defense fleets are defending.

If that's not a lot of flying fleets in total (less than 8), the AI looks for idle fleets to restock the front planets.

Who can't send fleets for restocking:

- front planets themselves
- planets whose value is less than the threat value of their orbiting fleet
- planets with orbiting fleet's threat level lesser than 1/25th of the empire's standing forces

I mentioned a couple of new things here:

- colonies' value for their owner — the same as the valuation of enemy planets, just with 5 points deducted for each missile base
- standing forces — the summary threat level of all fleets that were not flying anywhere at the last fronts calculation (i.e. after colony ships were sent off)

Everyone else donates their fleets to the front planets with the priority given to the weakest front planet at the last fronts calculation.

Well, a non-linear formula is used there including distance, but even if I'll explain it we'll come to practically the same result.

"Weakness" or "strength" of a front planet is the relations level with the target empire (can range from 100 to -100) plus the percentage of the total standing empire forces already stationed at that front planet.

If an empire has a front planet A with 20% of empire's forces and -20 relations with empire X and a front planet B with 30% of the empire's forces and -35 relations with empire Y, the front planet B is likely to get all the reinforcements.

Invasions and population transfers

As usual, the AI makes a list.

The colony is added to the invasion list if it:

- is orbited by ships of that AI
- is owned by a non-ally
- is colonizable
- and an invasion fleet of this AI is not flying towards it already

When targeting the human player the AI also checks for the ceasefire period, but when targeting other AIs it doesn't bother.

Now, as already should be familiar to you, another list is created.

This time it's the donor list and it's made out of colonies the AI has. If the colony's population is higher than half of the planet's current capacity and that capacity is higher than 40, the planet is conscripted.

The AI doesn't really plan invasions.

It just goes through the target list and for each target finds the closest donor that didn't send anyone this turn.

Then half of the donor's population is sent. That's all.

There are some distance checks, but they are relaxed enough to not matter most of the time.



Bolstering population works the same way, it's just the target list is also made out of this AI's colonies.

It happens right after the invasion transports are sent, and it makes sense. Some colonies may be missing half of their population now.

So, the recipients are colonies with the population less than 1/3 of their maximum capacity and those that have a rebellion.

Recipients that already have transports flying to them are ignored.

The donors are colonies with population number over 3/4 of their maximum capacity. 1/3 of the donor's population is sent to the target and the donor is dismissed from the pool.

No check is made whether the recipient can house this population or not.

Remember, the AI generally has no concept of ETA in turns, so projecting the free capacity of the target colony when this fleet will arrive is something that the AI just can't do.

Housekeeping

If any alien fleet is heading towards the colony (in "Classic+", non-allied) the AI overreacts and diverts all the colony's defence, industry and tech spending to shipbuilding, no matter what.

Also, 1/5 of the overall empire reserve is doled out to the colony.

Then for each of owned colonies with less than 20 population one fifth of what remains of the empire reserve at this point is given (it gets progressively smaller, of course).
It may get adjusted later.

And then the taxes are set.

For turns 1 to 19 the tax rate is 0%, since that the tax rate is 3-12% plus difficulty modifier, i.e. from 3-12% on simple to 7-16% on impossible.

The bug

That's all nice and dandy.

But that's not exactly Master of Orion 1 you played in the 90s and 00s.

This is how it was supposed to work, but it's not how it worked.

Master of Orion 1 does a very smart thing in its decision making.

When I was saying "close enough", I meant that these decisions are dependent on things being in 1 or 2 or 3 turns flight distance. It obviously scales with engine tech.

It is very smart and prudent to measure distances in turns or seconds it takes for an entity to cover them.

If I tell you that an arena in a MMO shooter is 1800 meters from side to side, it doesn't say anything.

It can be an overwhelmingly vast arena for snails with crossbows, it can be a really cramped arena for a WW2 dogfight.

But if I'll tell you the arena is 90 seconds from side to side, that's actually informative.

Everything's good, but there's a subtle bug:

```
static int game_ai_best_speed(const struct game_s *game_state, player_id_t player_id)
{
    int v = game_tech_player_best_tech(game_state, TECH_FIELD_PROPULSION, 0, 6, 50, player_id);
    if (game_state->ai_id != GAME_AI_CLASSIC) {
        /* WASBUG v is the engine tech level (0, 6, 12, ...) while caller expects speed (1, 2, 3, ...) */
        v /= 6;
    }
    return v + 3;
}
```

The function used in all these "close enough" calculations returns *engine tech level* + 3 instead of *speed* + 3.

It means the AI's reflexes are "wired" for much higher engine speed than it has.

An AI with nuclear engines behaves as if it had ion engines.

An AI with sublight engines operates essentially out of the design bounds.

Remember all that talk about fronts and front planets?

It will work mostly as intended with retros, but with nuclear engines the AI most likely will have only one front open.

It will launch attacks against every rival on the map from a single front planet placed who knows where.

Proper main attacks? Mostly gone.

Reinforcements from reasonable distances? Forget about them.

Having a visible sizable fleet where it makes sense? Hah, not gonna happen.

The AI is hollowed out.

It was hollowed out from the start.

This one-line change by Kilgore Trout on the picture above is the single most important AI fix in the game.

Of course, it's humbly made optional.

Thoughts

If the overall design wasn't so robust, the bug would've been caught in v1.1 or v1.2.

Instead, it persisted over the quarter of the century.

On the other hand, if the overall design wasn't so robust, Master of Orion 1 may have not become the classic we know and study.

Sure, after reading the logic overview you may think MOO1's AI is as smart as a unicellular organism.

It has reflexes, but it has no understanding of its surroundings. It can coordinate its actions only by coincidence.

And it has obvious, glaring areas where small changes may bring big improvements.

I mean, just replace all hull weight calculations by threat level and the AI will become much harder to beat. Not necessarily better, but harder.

Rewrite the threat level calculation to account for battle computers and it'll become even more guarded against any normal player.

It can be easy to decide it's a simple, primitive, bad AI, but it would be pretty far from the truth.

Yeah, with the powers of hindsight and modern tooling combined (our current tooling is exquisite) it's very simple to point out the errors made.

What's very hard is to point out what exactly in this design made it so captivating and inspiring even in its crippled state.

How did it persist? Why are people still playing the game? Why are you reading this?

It maintains the illusion oh so right with oh so limited capabilities.

I think it's downright inspiring.

More than that, I believe the framework of p1 phase can bridge gaps in your design.

If you struggle to make the AIs behave like proper opponents, just adapt the distance calculations to your terrain and copy the concept of front planets and different types of unit weighing.

With fairly minimal time investments you'll be able to see your system living and breathing instead of staring at the map on which nothing happens. It's hard to overestimate the importance of seeing your game running like a proper game for the first time.

Overall, MOO1 provides us a very adaptable and capable placeholder that one may replace later or improve upon or even leave parts of it as is.

After all, it was good enough for Master of Orion to be the Master of Orion.

Ship design (p2 phase)

Here's how it looks ([full res](#)):

It makes sense as, as you will see, at least one ship design will end up on the chopping block no matter what.

Scrapping ships

Master of Orion 1 allows a player to have just six ship designs at the time.

If you want a new design and your roster is already full, you've got to scrap one, but beware, all ships of that design will be scrapped too.

Most probably it was dictated by the UI limitations. When your canvas is just 320x200 pixels you have to choose between a lot of small sprites or a few big sprites. MOO1 designers obviously chose big ones.

While it can be (and is designed as) a conundrum for a human player, the AI simply deletes the oldest designs.

Before turn 100 it deletes designs that are older than 50 turns. After turn 100 it deletes designs that are older than 100 turns.

Then the AI checks how many designs with a non-zero number of ships it currently has. If there are 5 or 6 such designs, the oldest one is deleted.

If there are 4 or less such designs, the AI checks for designs with zero ships and deletes the oldest one again (technically, the last one by id, but designs are sorted by their age).

Now the AI has a free design slot.

Designing ships

Internally, AI recognizes three types of ships: colony ship, bomber and fighter.

The AI strives to have all these roles covered. If they are covered, the AI will design just fighters.

The AI never aims to build dedicated scouts because, as you surely remember, the AI is omniscient anyway and scouts just for show.

First the AI chooses a hull. Then it installs the best engines it has.

$\frac{1}{3}$ of the remaining free space is reserved for weapons.

When the AI tries to install something, it will check afterwards, did it encroach on the weapons space.

If it did, then it bails out of the design process and goes to weapons (but with a small special detour).

That's why the order in which modules are handled is very, very important.

If it's a colony ship, reserve fuel tanks and colony base will be added.

The AI will consecutively try to install the battle computer, shields and armor in exactly this order.

For a colony ship it's the end of the design line. It'll have a single laser added to it and declared good to go.
Other types go further.

Specials into slots 1 and 2 are chosen, maneuvering upgrades are added, jammer is added and the special for slot 3 is chosen.

Then some very special specials can be added to the ship, regardless of the weapons space left and weapons are chosen for the ship.

Pretty straightforward, but with a lot of interesting details we'll jump right into.

Choosing the hull

Colony ships always have large hulls.

Fighters and bombers have following probabilities:

Hull size	Rivals don't have pulsar tech			Rivals have pulsar tech		
	Normal	Alkari and Klackon	Bulrathi, Meklar and Silicoid	Normal	Alkari and Klackon	Bulrathi, Meklar and Silicoid
Small	8.3%	33.3%	0%	0%	25%	0%
Medium	25%	25%	8.3%	25%	25%	0%
Large	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%
Huge	25%	0%	50%	33.3%	8.3%	58.3%

Note how this table underscores the intended "look" of races.

Insects and birds get a fair chance at small designs, while bulky bears, cyborgs and living rocks tend to have lumbering ships.

Choosing armor

The AI always uses the best available armor for its ships as it's a natural thing to do. However, it'll install the double armor sometimes:

	Small	Medium	Large	Huge
Chance to get double armor	1%	2%	4%	8%

Choosing modules

Computers, shields, jammers and maneuvering all have different levels available to the player.

Min-maxing them in order to get the most value out of the ship is fun.

The AI doesn't have the luxury of designing ships intelligently.

Instead, when installing, let's say, shields, the AI counts available shield classes that can fit into the remaining space. Then it looks in the lookup table for shields and gets the probability for this hull. Let's say it's X.

With the probability of X% the AI will install the best shield it can fit. If the check fails, it'll install the second best with 2*X% chance. If it fails, the third best has 4*X% odds, etc. And that goes on until some shield is installed or the AI is all out of shields.

Now, what are these probabilities?

I'll go down to the fourth best, but the checks don't end there. I'll include just enough rows for you to easily understand the pattern.

<i>Battle computer</i>	Small	Medium	Large	Huge
Best	10%	25%	35%	50%
Second best	20%	50%	70%	100%
Third best	40%	100%	100%	100%
Fourth best	80%	100%	100%	100%

<i>Shield</i>	Small	Medium	Large	Huge
Best	5%	15%	40%	70%
Second best	10%	30%	80%	100%
Third best	20%	60%	100%	100%
Fourth best	40%	100%	100%	100%

<i>Maneuvering</i>	Small	Medium	Large	Huge
Best	30%	20%	20%	15%
Second best	60%	40%	40%	30%
Third best	100%	80%	80%	60%
Fourth best	100%	100%	100%	100%

<i>Jammer</i>	Small	Medium	Large	Huge
Best	2%	8%	20%	30%
Second best	4%	16%	40%	60%
Third best	8%	32%	80%	100%
Fourth best	16%	64%	100%	100%

Choosing specials

Specials are handled differently from other modules.

Each time the AI thinks about how to populate a special slot, it “constructs” a new table.

It takes a big table with probabilities, gets the column for this hull size, removes all specials it doesn't yet have, removes all specials that conflict with the ones already installed and ends up with a stub of the table, something like this:

Battle scanner	30%
Repulsor beam	25%
Inertial stabilizer	50%
Automated repair system	25%

Then it goes down the line, from basic specials to more sophisticated and rolls the dice.

Later successes supersede earlier successes, so if the AI rolled both battle scanner and inertial stabilizer, this slot goes to inertial stabilizer.

Here's the full table it operates with:

	Small	Medium	Large	Huge	Conflicts with
Battle scanner	0%	0%	30%	80%	
Anti-missile rockets	0%	10%	50%	80%	Zyro shield, lightning shield
Repulsor beam	0%	0%	25%	80%	
Warp dissipator	0%	0%	20%	70%	
Energy pulsar	0%	5%	20%	60%	Ionic pulsar
Inertial stabilizer	20%	30%	50%	80%	Inertial nullifier
Zyro shield	0%	10%	40%	70%	Anti-missile rockets, lightning shield
Automated repair system	0%	0%	25%	50%	Advanced damage control
Stasis field	0%	0%	20%	40%	
Cloaking device	20%	30%	40%	70%	
Ion stream projector	0%	0%	20%	30%	
High energy focus	10%	20%	25%	35%	
Ionic pulsar	0%	0%	20%	30%	Energy pulsar
Black hole generator	0%	0%	20%	40%	
Subspace teleporter	10%	20%	30%	40%	
Lightning shield	0%	10%	30%	50%	Anti-missile rockets, zyro shield
Neutron stream projector	0%	0%	30%	60%	
Advanced damage control	0%	0%	0%	50%	Automated repair system
Technology nullifier	0%	10%	20%	30%	
Inertial nullifier	10%	20%	40%	60%	Inertial stabilizer

Oracle interface	10%	20%	40%	60%	
Displacement device	10%	20%	40%	60%	

It's a fairly powerful instrument.

It all but ensures that large designs will have some of the best specials and chances are, you won't like it.

On the other hand, it allows for a lot of controlled variability in the designs.

The only issue I see here is that it underestimates high energy focus very significantly.

Special detour

A couple of specials may be added to the ship regardless of weapons space left.

If slot 1 is empty, AI's propulsion tech is level 30 or higher and the ship doesn't have an inertial stabilizer or inertial nullifier, it'll get an inertial stabilizer into slot 1

If slot 2 is empty, the ship is large or huge and it has no battle scanner, it'll get a battle scanner into slot 2.

Weapons selection

The game selects how many different weapons the design may have depending on the hull size.

	Small	Medium	Large	Huge
1 weapon	100%	50%	25%	0%
2 weapons	0%	50%	37%	0%
3 weapons	0%	0%	29%	50%
4 weapons	0%	0%	9%	50%

The AI also randomly decides whether to use 2-shots missiles or 5-shots missiles for this design. It's fifty-fifty.

For each slot the AI selects the best weapon it has, but hasn't yet installed on this ship.

So, if the AI wants to use three slots, it'll use three best weapons.

It'll go down a notch if it's building a fighter and the first weapon happens to be a bomb.

Now, the AI goes slot by slot and calculates how much of these weapons it can fit into the remaining space, just like a human would do.

And then it starts to cheat.

Royally.

Let's discuss the simplest variant when the design is meant to handle just a single weapon.

The AI rolls the dice and installs from 100% to 141% of weapons it's allowed to install. Granted, we're still in the realm of integer division. But if the AI was meant to install just 3 gatling lasers, with a lucky roll it can install 4.

Overall the percentages seem to go like this:

	slot 1	slot 2	slot 3	slot 4
1 weapon	100-141%			
2 weapons	60-121%	100-141%		
3 weapons	40-111%	40-111%	100-141%	
4 weapons	40-101%	40-101%	60-121%	100-141%

Of course, if the AI goes overboard in slot 1, it won't have space left to use on subsequent slots, but that's of little solace.



(Typical early-mid game Silicoid ship)

Bloopers

The AI tries really hard to design ships.

But as the design method is fairly random, the *game_ai_classic_design_ship()* function is called in a loop that can theoretically run up to 10000 times.

Don't clutch your pearls yet.

The core logic of the design process in that function resides in two nested loops and can theoretically run up to 100000 x 100000 times.

So, in total, the design process can run up to 10000 x 100000 x 100000 = 10¹⁴ times.

That's a hundred quadrillions and now it's time to reach for those pearls.

Luckily, the game manages to design something suitable much, much faster, usually on the first or the twelveth or the two hundred thousand and fifth try.

The AI doesn't really care what the other empires field.

Each AI kinda does its own thing in design with two exceptions.

First, the AI cares if anyone it knows on the map has energy or ionic pulsar tech. It considers this tech to be the bane of small hulls, so it adjusts its hull priorities a bit. That's fine.

Second, the AI cares if anyone has repulsor tech or warp dissipator. In the AI's opinion, that makes weapons with the range of 1 obsolete. That I see as a flaw of the algorithm.

Realistically, the AI can loop through 100000 design attempts and achieve nothing if someone knows how to make repulsors and:

- the AI doesn't have range 2 weapons (this is probably why heavy laser is a starting tech)
- the AI chose a small hull for the design attempt and heavy weapons won't fit

In the latter case the AI just tries again, in the former the AI kinda says "screw it" and removes the requirement for at least one range 2 weapon in the design.

The correct response to repulsors is, I believe, pushing maneuverability up in the order of module installs, making HEF more desirable to the AI and checking for range 1 only if speed equals 1 and HEF is absent.

Then again, what I deem incorrect worked for the quarter of a century and granted a lot of players a lot of fun pushing slow bombers away from their colonies.

The AI will also invalidate the designs that have:

- missiles or bomb as slot 1 ("main") weaponry and have high energy focus special
- missiles as slot 1 ("main") weaponry and have oracle interface special

Building stargates and ships

At the end of the design phase designs are sorted by their age.

The newly constructed design obviously gets the first place.

Then each colony is randomly assigned one out of the three newest designs to build, unless the planet needs to build a stargate.

Now it's off the p2 phase hands and it can't care less when the ships will be built as colony spending is the p3's phase business.

Frankly, building designs shouldn't have been a ship designer's job anyway, but saying "yeah, any one out of the three newest designs is okay" is a bit disappointing.

But I mentioned stargates.

In MOO1, stargates are the special seventh ship design, so building them is handled here as well.

It's very simple, if the empire has the tech and this particular planet doesn't have a stargate, with 50% probability it'll start building one. Rinse and repeat for every colony.

The AI doesn't care about how long it'll take the colony to build a stargate and it doesn't care about maintenance costs either.

There is a potential for AI to bog itself down, but it's quite uncommon for an AI to live long enough to enjoy stargates.

Thoughts

The ship generation is... okay.

I expected horrors when I first looked at the 10000 * 100000 * 100000 loop and I didn't find any.

The most convoluted thing I saw was `v &= ~1;` and that's pretty tame. Aside from weapons placement I didn't even see any cheating.

As the p1 phase AI can't communicate any intent to the ship designer, it just makes an okay ship and can accidentally catch the player off-guard.

I largely agree with the ship design choices when the task is framed this way and I like how it ensures that more or less modern parts for the empire will be used.

However, there's still a lot of room for improvement in the framing of the task. The AI just does it because it does.

Sure, the p1 and battle parts of AI aren't smart enough to deal with more specialized ships in a more sophisticated manner, but even the existing roles could be utilized better.

Yes, even back in the day.

With the power of hindsight, I declare!

Colony ships, as the AI cheats with them, may be redesigned the moment the new engine tech becomes available, not when the old colony ship design gets deleted in the end.

Bomber designs easily can prioritize maneuvering and anti-missile defences as the first special. Also, they don't need three or four slots.

The existing weapons placement logic will work very well for them with just two slots for medium and larger hulls.

Most importantly, there's no need for an AI to scrap a design before attempting a new one. These horrendous loops were made out of the necessity to fill the hole with something, anything that's valid.

An AI may prepare a new design, compare it with the existing ones by the combat value and then decide whether it wants to delete something or not.

This way the AI can try just a hundred times or so and if nothing good came out of it, no big deal.

That said, I can quite recommend this approach as a placeholder, especially the module selection parts.

You'll have to rethink your approach to ship design once you understand better how your game plays, however.

If you decide to go through roles (or even tagging the designs with various roles), both your p1 and battle handling analogs should be capable of dealing with all these roles plausibly.

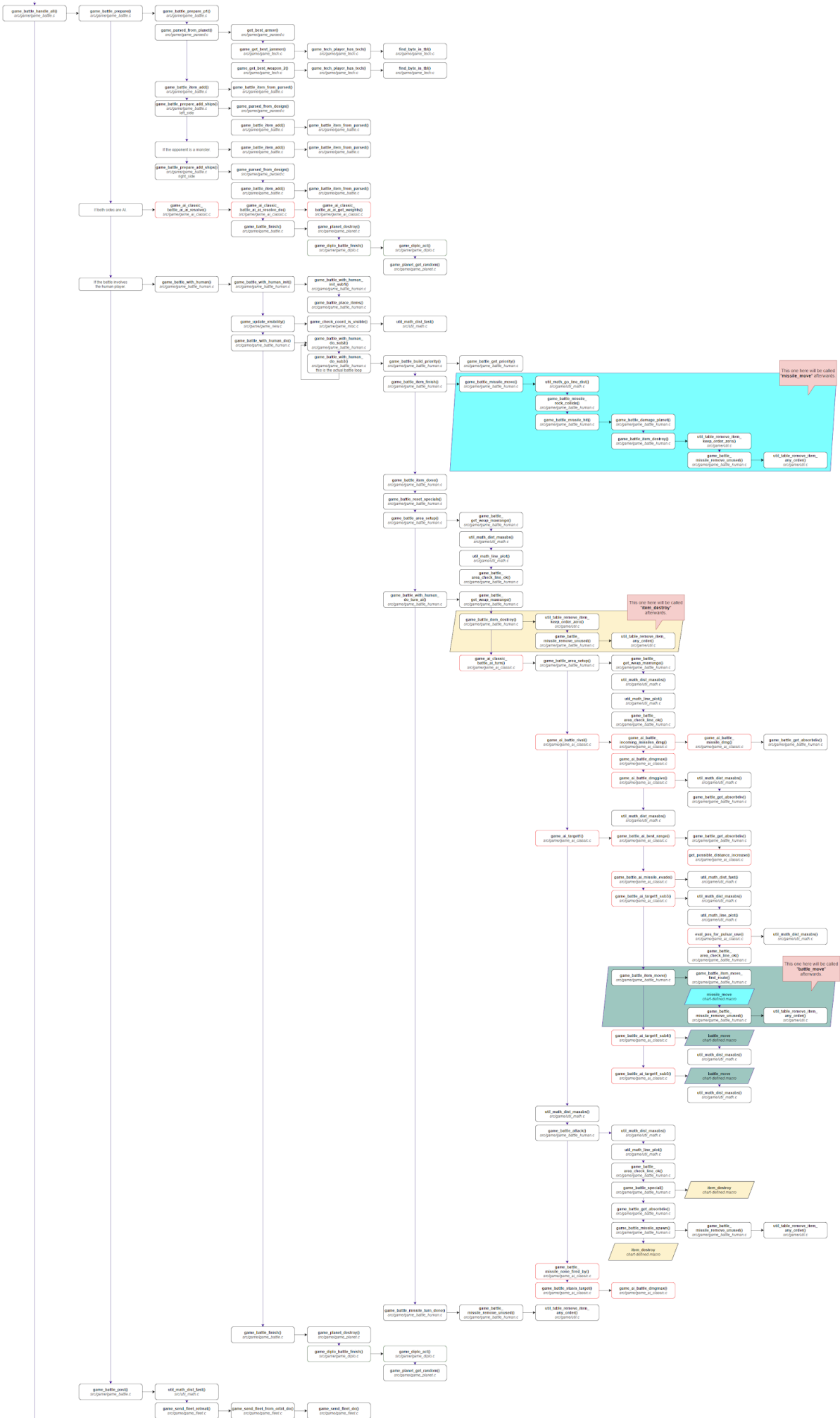
Ideally your AI should be able to understand why and how another player beat it in combat and adjust designs accordingly or beg for peace if it thinks it can't counter.

Roughly half of the heavy lifting in this case can be done through the mock-up battles the AI may “perform” with known enemy designs. But there’s about two freight cars of work in this part only.

Battle handling

While you still remember how ships are designed, let’s postpone the p3 phase and hop over to battle handling.

Here’s how it looks ([full res](#)):



That's scary.

Luckily, a lot of it is code related to the human interaction with the battlefield, missiles handling, etc.

Don't worry, it's much simpler than it looks like.

AI vs AI

Outcome

How do AIs battle when no one's looking?

Poorly.

They "trade" simulated blows until one side dies.

It's a good thing we as players don't see that.

Imagine you and I are battling this way.

You have 100 points, I have 100 points.

You roll the dice up to 10, I roll the dice up to 10.

I'll be generous and assume you rolled 8 and I rolled 5.

Now you have $100 - 5 = 95$ points, I have 92 points.

You roll the dice up to 10, I roll the dice up to 10...

And that goes on until someone reaches zero.

Let's assume I lost.

I have zero points and you have 20.

If we were MOO1 AI's, now I'd nullify my fleet and you'd leave 20% in each stack in yours.

That's how AIs battle.

We started fair, with both sides equal, like in a tactics game.

That's purely for the ease of explanation.

Strategy games are all about making the battle unfair, be it through quantity (production, logistics, comms) or quality (tech, designs).

So AIs don't have equal powers, but calculate them at the beginning of the battle.

Then they trade blows worth 1-10% of their initial power each combat round. Note, initial, not current.

The actual percent is random, but if the blow turns out to be less powerful than 1, it's set to 1.

The blows are deducted from their current powers and the next round begins, just like described above.

Generally speaking, this means that if you have twice more power than me, most likely you'll get away with 75% of your fleet remaining. Thrice more power than me, delete ⅓ of yours. Four times... you get the idea.

Counting

Remember I described two ways the AI calculates the combat power of a ship and teased the third?

Well, now it's time for that third.

It's somewhat close to weight by hull multiplied by threat level, but even higher.

It can get so high that it was the source of a well-known bug in the original MOO1.

```
/* WASBUG MOO1 uses 32bit math which results in negative overflows with large ws2,  
| giving negative ship counts which get limited to 32000 later.  
*/
```

More specifically, the weight by hull (1 for small, 5 for medium, 25 for large and 125 for huge) is multiplied by the sum of:

- 5 * shield level (level, not class, remember?)
- 5 * battle computer level
- level of the primary weapons (the ones in the first slot)
- 10 * armor level
- sum of specials' levels multiplied by 2
- ½ of construction tech field level
- 15 (20 in Classic+) if the ship is made by Mrrshan or Alkari

Note that the number of primary weapons is irrelevant.

HP is also not mentioned, doubling the armor will give the AI just one additional point in armor level.

And maneuvering is, again, missing in action.

It kinda works because all AI ships are made using the same priorities and assumptions.

If you have the construction tech level and hull size, you can estimate the space left for weapons in the first slot.

Given the level of the weapon and the fact that it'll be the best or the second best one, you can understand roughly how many of them there are.

The construction tech level used is current though, so older designs may be slightly overestimated by the AI.

But the AI designs don't live that long, so the effect is negligible.

But against a human... yeah. A different design approach is needed.

If a colony with missile bases is present in the battle, it gets counted too.

The weight of a single missile base is $75 * (\text{weapon tech level} + \text{shields tech level})$.

While the formula is meant to represent missile tech and planetary shields, no check is made whether the AI has them or has upgraded the colony defenses to them.

In case a monster attacks the AI, it gets assigned combat power of 30000.

Curiously, the case when AI attacks the Guardian or vice-versa is accounted for too. To the best of my knowledge can't happen, but the Guardian has the combat power of 100000 still.

AI vs human

So, what happens in a battle that involves a human player?

Quite a lot and everything worth mentioning from the player side is described in "Master of Orion Strategy Guide" by Emrich and Hughes in chapter 7 (seriously, can't recommend it hard enough).

The order of initiative, how missiles interact with asteroids, etc. A lot of good and interesting player-facing things.

But we're interested in what's behind the curtain, right?

Each ship stack wages a war of its own, with awareness where the friendlies are, but with no regard to what they are doing.

I'll describe just the logic, because there's a lot in there.

If you're particularly interested to look for yourself the details of the implementation, [`game\_ai\_classic\_battle\_ai\_turn\(\)`](#) is the place to go.

I warn you, though, it's a bit of a minefield.

Some things are passed indirectly through the battle state structure, some functions do things on the battlefield and yet return a value... it's a pretty hard code to follow.

High-level logic

So, a stack needs a target.

There are three modes to find a target: without moving, without moving including stasised enemies and with moving.

If there's no valid target in any mode (and rules are pretty complex for this), the stack retreats if it can (it's not stasised and there are no missiles flying at it).

The planet is also a stack, so in this case it shoots at any enemy even though it very well knows the attack is futile.

Generally speaking, the stack first searches for the best rival it can tackle when moving.

The stack calculates its best range, i.e. the range at which it'll deal most damage. Weapons, specials, everything is accounted for.

If there's a target and the stack isn't attacked by missiles, it scans the battlefield and finds the best position it may take, including checking for a place where to shoot energy or ionic pulsar.

Then it moves to it. During the stack's move all missiles that are aimed at it also move. Also it may be subject to a "retaliation" attack if it flies near the enemy and this enemy has unspent shots.

Then the stack attacks. Pew-pew, missile swarms, damage indication, target destruction, etc go on.

Maybe the stack is repulsed in the end by a surviving enemy with a repulsor or even destroyed in a retaliation.

If the stack was not destroyed and still has weapons to fire, it searches for the best rival it can attack without moving and attacks it.

This can loop up to ten times in the code, but, practically speaking, four weapon slots, one used, three specials... probably up to six. Theoretically.

Some megadeathstar is needed to kill so many targets while standing still.

There are two caveats.

If the stack isn't attacked by missiles, but there's no valid target even when moving, the stack will move to the closest opponent.

If the stack is attacked by missiles, it will go into the evade routine.

That's probably it.

As I said, this code is fairly hard to follow, so I apologize if I missed something somewhere.

Distance calculation

Oh, that's easy.

The battlefield in MOO1 is a 10x8 square grid.

The game uses distances in these squares for combat logic (except for missiles positioning).

Now, the distances between As, Bs and Cs you can see below in the grid are all the same for the game, all are at 3 squares distance.

	0	1	2	3	4	5	6	7	8	9
0										
1		A			A					
2										
3		B						C		
4										
5										
6					B					C

7										
---	--	--	--	--	--	--	--	--	--	--

The game simply looks for the difference between x coordinates, between y coordinates and returns the larger one.

Finding a rival

The game pulls out all stops when searching for the best rival for the stack.

It takes into consideration:

- the maximum damage that this stack can deal to the target (here's where the modes spring up)
- the maximum damage a ship in the target can deal (including damage from bombs and bio if the planet is on the AI's side)
- how many ships in the target stack will die from the missiles already targeted at it
- how much HP can the target repair in a turn (relevant for large and huge ships)

A planet has two modes of attack, normal mode and MIRV mode, so it checks in which case the damage will be higher.

Now, the resulting weight, the priority of this target can be 0, 1, 2, 3, hundreds, thousands and a lot.

Not kidding.

Generally the weight of the target is the expected number of kills multiplied by the damage of a target ship.

Makes sense, you want to get the most impactful ships off the board as quickly as possible.

However, there are a lot of nuances.

If a stack can deal bio damage to a planet and that damage overwhelms the antidote tech of the defender, it's highly prioritised.

The weight is $100 * \text{effective bio damage} * \text{number of bio bombs}$.

If an attack can kill more than 20 ships in the target stack, a very strange check is made.

The game imagines all missiles that are currently flying to the target hit all at once.

Then it estimates the number of ships left in the target stack and compares it to the number of kills the attack can make divided by 20.

```
after_missiles_num = target_stack->num - missile_victims_in_target_stack; /* FIXME BUG? num - dmg? */
if (after_missiles_num < frags_by_twenty) {
    frags_by_twenty = after_missiles_num;
    SETMAX(frags_by_twenty, 0);
}
```

So if the attack could've killed 72 ships, but the incoming missiles will leave 17 ships flying, everything works as it's supposed to be.

The weight will be max damage from target multiplied by 20 and by 3 (because there are only three twenties in seventy two).

But if the incoming missiles will leave just 2 ships standing, then that max damage by 20 will be multiplied only by 2.

If the attack will kill less than 20 ships and the target stack already has missiles flying at it and this stack can deal some damage to it, the weight of this target equals... drumroll... 3.

If the weight of a target equals 1 and the target happens to be a planet, it gets weight 2.

A planet adds 7500 weight to every target it evaluates, but deducts 750 for each square of distance.

It's a very strange mix of ultra-low and ultra-high weights.

It feels like it was imagined as a straightforward "how much enemy damage can I delete by this attack" and then a lot of ad-hoc measures were added to "fix" this or that corner case.

Damage giving and dealing

The amount of damage a target stack can deal is calculated pretty intuitively.

Get all the weapons, get their max damage, multiply by the number of shots in a salvo and you get the alpha strike.

If the weapon is a bomb, the damage of it is either 0 (no planet in this battle or this stack is defending) or max damage minus planetary shield absorption.

If the weapon is a bioweapon, the damage of it is either 0 (for the same reasons as bombs) or 50 times its effective bio damage (bio damage minus the target empire's antidote efficiency).

If there's a black hole generator in that stack, well, that's plus 1000 weight, no questions asked.

The amount of damage the AI's stack can deal is even more human-like.

It takes into account shield absorption, shield-penetrating weapons, the reduction of non-bomb damage to planets by 2, etc. Everything a knowledgeable player would do.

Interestingly, AI calculates the best possible and the worst possible damage of a weapon separately. Then it averages the results and goes on with it.

If a weapon is bio, its weight is the bio damage multiplied by 500, regardless of on which side the planet is (and is it on the battlefield at all).

Now what do specials bring to the table from the AI's point of view:

- warp dissipator — 100 damage points
- black hole generator — 1000 damage points
- technology nullifier — 500 damage points
- energy pulsar — (5 - target shield absorption) multiplied by the target stack size; target stack, not attacking stack and it's probably a bug
- ionic pulsar — same as above, just with 14 base damage

- ion stream projector — worth $(20 + \text{attacking stack size} / 2)$ percents of the cumulative HP of the target stack
- neutron stream projector — same as above, just with $(40 + \text{attacking stack size})$ percents

Calculating the optimal range

The best range for an AI to attack a planet is 1. As close as possible.
End of story.

If the target is not a planet, the AI cycles through all ranges from 1 to 10 and estimates the damage output.

There's a lot of things going for range 1:

- black hole generator adds 2000 weight
- stasis adds 2000 weight
- energy pulsar adds $(6 - \text{target shield absorption})$ multiplied by the target stack size
- ionic pulsar adds $(16 - \text{target shield absorption})$ multiplied by the target stack size

Range 2 also has its supporters:

- ion stream projector adds $(20 + \text{attacking stack size})$ percents of the cumulative HP of the target stack plus 0.99 of the target stack size
- neutron stream projector adds $(40 + \text{attacking stack size})$ percents of the cumulative HP of the target stack plus 0.99 of the target stack size

Hey, I'm not the one who came up with these formulas. I'm just trying to explain them in a human language.

Now the damage for weapons is calculated as usual. Maximum damage minus absorption, multiplied by... you know this stuff.

Nuances are:

- high energy focus is accounted for in here
- the estimated maximum movement of the target in two turns is added to the usual missile range (i.e. how far it'll fly in one turn)
- weapons that have their damage fade (various torpedoes) are accounted for in damage calculation only at range 1 (they still will be fired from larger range, just not accounted for)
- bio bombs are worth their max damage multiplied by 100 in this case

Range 1 is very likely to turn out to be the winner in the comparison.

So the game looks for the pure maximum range with maximum damage only when the stack has cloaking.

Usually it searches for the maximum range that still retains at least two thirds of the range 1 damage.

Miscellaneous

If a stack has the stasis generator, it looks for someone to stasis before everything else. However, the AI won't stasis the last two unstasised enemy stacks.

The selection of targets is simple.

The AI ignores cloaked ships and already stasised ones and will go for the stack that can deal the most damage.

When deciding if it's a good position to use a pulsar (energy or ionic), the AI simply looks at squares around it.

If a square holds a friendly stack and the pulsar blast can get through its shields, the AI deducts 4 points from the value of the position. If there's a damageable enemy stack, the AI adds 3 points.

The AI doesn't check for the HP of targets.

Evasion is simply trying to get as far away from the missiles as possible. Well, in theory.

Retaliation is simply an attack of the attacker if the stack can retaliate (i.e. didn't retaliate this round and the attacker is in-range without moving).

Retaliation may include repulsion if the defender has repulsors.

Pathfinding and area scanning... eh, it's of little use for modern development.

Pretty interesting old tech, but not much more.

I mean, fitting the scan of a 10x8 grid into an array of ten 8-bit integers is very cool, but I can't imagine a situation in which it'd be valuable today.

Maybe some MCU programming, but surely not game design.

Thoughts

I like the combat in Master of Orion 1 (and I obviously like the game very much) and understanding the underlying concepts of it makes my mind wonder.

The number of what I perceive as fixes to the original idea is a bit alarming.

I read them as an indication that the simple and straightforward initial idea turned out to be a little bit too simple and straightforward.

I believe the original design ran into the realization that different ship types (especially the ones made by the human player) should be viewed in a different context, so that reaction to them is believable.

The context is not only the design of the probable target, but also the design of the attacking ships and the goals of belligerents in the battle.

The ad-hoc fixes helped the combat AI to be very believable in the confines of what MOO1 can offer, and that's great.

What's not great is that your game is likely to have different design priorities in mind, so copying the MOO1's design may be not that suitable even for a placeholder.

However, the underlying assumptions of it are very sound.

The battle AI is made to be reactive to the player's actions.

Sure, there are some very heavy-weight goals like bombing that colony into oblivion.

Still, the player's choices before and during the combat define how the combat will go, not AI's actions.

I'm inclined to believe it's a good thing.

This way it's easier for the human player to learn to trick the AI and feel good.

I'll run a small thought experiment here.

Imagine we're using mostly the same routines, but the AI does not know what's inside of enemy ships. So it's kinda forced to use hull weight multiplied by stack size to know which ships to attack first.

Then the player's ships open fire and the AI can now know the weapons used, reassess the combat value of ships and act accordingly.

We've just unlocked a lot of tactical depth for the player: when do you want to reveal your hand?

That gets boring after several fights, so let's allow the AI to have a bit more information.

First of all, let's allow it to gather which sizes of ships inflicted most damage to the AI so it'll prioritize the player's ships by size better.

The thing is that players tend to stick to a preferred size and the AI "responding" (not really) to that will be seen as an analysis.

Second, let's allow the AI to store the data about ships it already met in combat and to use espionage information (partial and years late, of course) in combat.

Now the AI will react properly to known designs and "cautiously" (not really) approach new designs.

We just gave the player the incentive to trick the AI, maybe even with inferior or one-shot designs.

But that'll make the AI too predictable, so let's introduce basic goals.

In MOO1 they are: "kill enemy fleet", "defend colony" and "destroy colony". We need that for the third thing.

Third, without any overarching AI, let's allow each stack to decide whether its losses are worth the goal.

As in "in this attack I'll lose three huge ships out of seven which is 6% of the empire's firepower and this colony is just 2% of the empire's production, I'm out."

Such a noble selfish retreat will both provoke a domino-effect in this particular battle and make a lot of AI's forces fall back (probably to a front planet) and "regroup".

The math doesn't need to be ultra-correct.

The player will feel "oh, they fear me" anyway and, with lower or equal warp speed to the AI, will feel unable to follow. "Oh, that planet is a tough nut now, maybe I should work on a cease-fire with them and hold what I got".

With a higher speed an aggressive player will just rip the heart of this front out of the AI's empire and, again, feel good about it.

This concept is not that straightforward to implement in MOO1, but from the technical perspective, not the design perspective.

The trickiest part is to add WEAPON_UNKNOWN into the enum of structures holding the weapons data without breaking a lot of subtle things relying on the exact order and numbering.

That, and a lot of plumbing from espionage to combat handling.

Everything else, I think, holds and can be neatly assigned to different difficulty levels.

That's a good thought experiment, I like it, thank you, thank you, but, probably, not that good time investment.

I'd better recommend to play the best turn-based designs and get inspired by them.

And by "inspired" I mean take everything you like that's not bolted down.

"Inspiration" recommendations

I believe Horn of the Abyss mod for Heroes of Might and Magic 3 has the best stack-based turn-based tactical combat at the moment.

It's steeped in about twenty years of competitive play and steered by people who play the game way too much.

It has incredible depth despite its seeming simplicity and a fantastic amount of interplay between spells, units, positioning and so on.

The drawbacks are that the units are fixed and the balance between different castles is non-existent. Some units and whole units lineups are plain better than another.

In a competitive 1v1 setting the balance is achieved through the pre-battle bargaining, who takes the better castle from a randomly selected pair in exchange for how much gold (who moves first is also bargained for).

I believe Rogueteck mod for Battletech has the best non-stack turn-based vehicular tactical combat at the moment.

I'd like to underscore the gradual reveal of what exact weapons an enemy mech has.

You know the number of weapons in each category, but you don't know if it is a SRM2 or a LRM20(C) until it fires at your guys.

More than that, the player can switch ammo or operation mode in-battle for a lot of weapons, adding to the uncertainty.

The need to constantly maintain evasion of all except for very tanky or overleveled (think 4.5 skulls lance at a 2 skulls mission) units by moving them around all the time is very worthwhile to contemplate too.

The drawbacks are that the mechanics kinda require terrain to work and scale pretty badly.

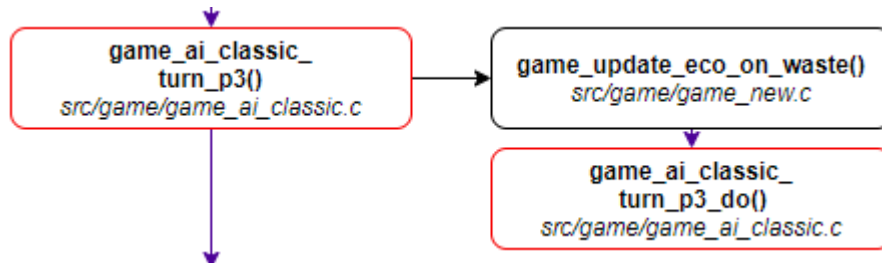
In MOO2 you at least do not care about positioning and hit chances that much.

And the amount of attention each mech in your company requires both in and out of battle is approaching the level of a party-based CRPG.

Just keep the pace brisk and the combat resolution quick. I've yet to meet a player who doesn't autoresolve tactical battles in Endless Legend.

Colonies spending, research, eco (p3 phase)

Here's how it looks:



Looks fairly small and self-contained, right?

Well, it handles a lot:

- planetary spending adjustment — first adjusted at 15-20th turn (or at 25-30th if an empire happens to not to have any colonizable planets in-range) and then each 2-6 turns
- handling overproduction — each turn
- research spending adjustment — adjusted each 10 turns, first adjusted at the turn of the AI's number in-game (1-5)
- spies mode of operation and security spending — each turn

Overall it looks like a mish-mash of high-level housekeeping functions.

You know what?

I'll describe spy-related logic from this phase in the spying overview and instead talk a bit about planetary eco adjustment. Okay?

Planetary spending

As you know, each colony in MOO1 has five sliders that represent how the raw productivity of the colony is used:

- ships — shipbuilding
- defense — building and upgrading missile bases and planetary shields
- industry — building new factories for more production
- eco — cleaning up industrial waste and generating new population
- tech — generating research points

The function that updates eco spending gets called every turn and sometimes it's called twice, when the planetary spending adjustment is also run.

Its logic is fairly simple: get the cost of waste cleaning this turn and if the colony spends on eco less than it should, proportionally reduce all other spending to eek just enough eco spending for the ecology to be stable.

In the context of adjusting planetary spending, this function defines the budget for ships, defence, industry and tech.

Bugs aside, AI can't spend on them more than $(100 - \text{eco})$ percent of the colony's output.

From this eco value the AI makes some baseline assumptions before applying planetary and racial effects:

<i>Eco baseline</i>	Ship	Def	Ind	Eco	Tech	Unassigned
If eco ≥ 96	0	0	0	96 - 100	0	0 - 4
If eco ≥ 91	5	0	0	91 - 95	0	0 - 4
If eco ≥ 81	5	5	0	81 - 90	0	0 - 9
If eco ≥ 71	5	5	10	71 - 80	0	0 - 9
If eco < 71	5	5	10	0 - 70	25	-15 - 55

Note the priorities of the AI.

Even if the colony is struggling to keep the population alive, shipbuilding and defense spending always come first.

The sliders' sum goes higher than 100 in some cases, reflected here as a negative number of unassigned points.

Don't worry, the regular eco spending call at near the end of the turn will fix that.

Now going with all these eco variations is going to make the following table harder to read. Let's assume we have eco at 30, a pretty normal value for early mid-game.

<i>Planet + race</i>	Ship	Def	Ind	Eco	Tech	Unassigned
Baseline	5	5	10	30	25	25
Planet has less than 3/4 max population	5	5	50	30	25	-15
Planet has artifacts or is Orion	5	5	10	30	45	5
Planet is rich or ultra-rich	15	5	30	30	25	-5
Owner is Bulrathi	5	5	10	40	25	15
Owner is Sakkra and colony sent transports	5	5	10	40	25	15
Owner is Klackon and colony sent transports	5	5	10	45	25	10
Owner is Meklar	5	5	30	30	25	5
Owner is Psilon	5	5	10	30	45	5
Owner is Mrrshan	20	5	10	30	25	10

The changes between various races are drastic.

We have four races with permanently skewed spending, four with normal spending and two with conditionally skewed spending.

Only unassigned points can be spent according to the traits an empire has

As the developing colonies ramp up their industry and waste, they stop being able to act according to traits and develop more or less like every AI's colony does.

The worst off are developing colonies of Pylon and Meklar, they have -35 points in debt. This means even in the very late game with zero eco costs they still have -5 unassigned points and just can't afford traits.

Let's talk about the lucky ones, the ones with 25-55 unassigned points.

The AI taps 2-10 points from this reserve and semi-randomly adds them to this or that field. That goes on until all unassigned points are spent.

Semi-randomly means that while AI empires have traits and, therefore, priorities, the game just rolls the dice each time.

Any given planet may end up with any points distribution, but with a large enough sample it's bound to follow a pattern.

This pattern:

<i>Traits effect</i>	Ship	Def	Ind	Eco	Tech
Diplomat	7,7%	15,4%	30,8%	7,7%	38,5%
Militarist	30,8%	7,7%	15,4%	7,7%	38,5%
Expansionist	15,4%	30,8%	7,7%	7,7%	38,5%
Technologist	7,7%	15,4%	23,1%	7,7%	46,2%
Industrialist	7,7%	15,4%	30,8%	7,7%	38,5%
Ecologist	7,7%	7,7%	15,4%	30,8%	38,5%
War	30,8%	7,7%	7,7%	7,7%	46,2%

There seems to be no difference between diplomat AI and industrialist AI from the spending perspective.

War is a special trait that all empires assume once they enter a war (so, a significant chunk of playtime).

So, our nice baseline planet will turn out roughly this way (averaging over a sufficiently high number of samples, of course):

	Ship	Def	Ind	Eco	Tech	Unassigned
Baseline	5	5	10	30	25	25
Diplomat	7	9	18	32	34	0
Militarist	13	7	14	32	34	0

Expansionist	9	13	12	32	34	0
Technologist	7	9	16	32	36	0
Industrialist	7	9	18	32	34	0
Ecologist	7	7	14	38	34	0
War	13	7	12	32	36	0

The sum of eco and tech is more or less constant for all empires except for the ecologist on the whole range of initial eco spending.

The less the eco is, the more pronounced the differences between different traits become.

We're left with only overbuilding to cover.

If a colony already built all factories its population can operate, the industry points are split between defense and tech.

If a colony already has a shield and too many missile bases, the defense points are sent to ship building (well, in MOO1 ship points were simply doubled due to a bug).

“Too many” missile bases is “at least 5” and the additional condition depends on difficulty:

- simple — more than population count divided by 15
- easy — more than population count divided by 12
- average — more than population count divided by 9
- hard — more than population count divided by 6
- impossible — more than population count divided by 3

Research spending

The research spending follows the same approach as colony priorities, as in, get the baseline, adjust by race, adjust by empire traits, but much more roughly.

The AI always devotes 75% of research to one tech field and per 5% to all others. In total the AI researches techs at 105%.

The baseline assumptions go like this:

	Probability	Computer	Construction	Force fields	Planetology	Propulsion	Weapons
Baseline	N/A	5	5	5	5	5	5
If turn < 30	50%	5	5	5	5	75	5
Bulrathi	25%	5	5	5	5	5	75
Mrrshan	25%	5	5	5	5	5	75
Darlok	25%	75	5	5	5	5	5
Meklar	12.5%	5	75	5	5	5	5

Meklar	12.5%	5	5	5	75	5	5
Alkari	25%	5	5	5	5	75	5
Sakkra	25%	5	5	5	75	5	5

If AI has got caught in any of those checks, it doesn't adjust the spending further.

However, likely it will proceed and the new focus (for these 10 turns) will be selected with the following probability:

	Computer	Construction	Force fields	Planetology	Propulsion	Weapons
Diplomat	12,5%	12,5%	25,0%	12,5%	18,8%	18,8%
Militarist	18,8%	6,3%	25,0%	12,5%	6,3%	31,3%
Expansionist	6,3%	6,3%	18,8%	31,3%	12,5%	25,0%
Technologist	25,0%	12,5%	18,8%	12,5%	12,5%	18,8%
Industrialist	6,3%	31,3%	25,0%	6,3%	12,5%	18,8%
Ecologist	6,3%	12,5%	25,0%	6,3%	31,3%	18,8%
War	12,5%	12,5%	25,0%	12,5%	6,3%	31,3%

That's how AI researches things.

Well, at least diplomat AI and industrialist AI differ here.

The choice of the tech to research next doesn't happen here (it happens in *game_ai_classic_tech_next()*), but it makes sense to describe it while we're at it.

With a certain probability the AI will choose the highest level tech in the field.

This probability depends on the difficulty level, it's 0 on simple and easy, 1/6 on average, 2/6 on hard and 3/6 on impossible.

Otherwise it'll just pick a random tech to research.

Thoughts

The colony development routine does a decent job for what it's supposed to do.

It wildly differs from what the human player usually does.

It's mindless, impassioned, geared for the long run and does not really care (aside from the war "trait") about what's going on.

It works, but it works because the player doesn't have an insight into what's going on.

The only way a player can see previous spending values is after capturing the planet, and, as we saw in p1, a planet starts cranking ships out in fear once any non-allied fleet is headed towards it.

It can work for you too, as long as you don't show it to the player.

Still, the more complex system you have in mind, the less MOO1's approach is applicable.

Research feels more like a rush job.

A bit of race-specific checks are thrown in, but overall it works just because the player has even less insight into research spending of the AI.

A more human-like approach would be to estimate the turns left to a tech breakthrough and decide through traits like: "no, I want that tech in 10 turns more than this one in 3 turns, because it appeals to my trait".

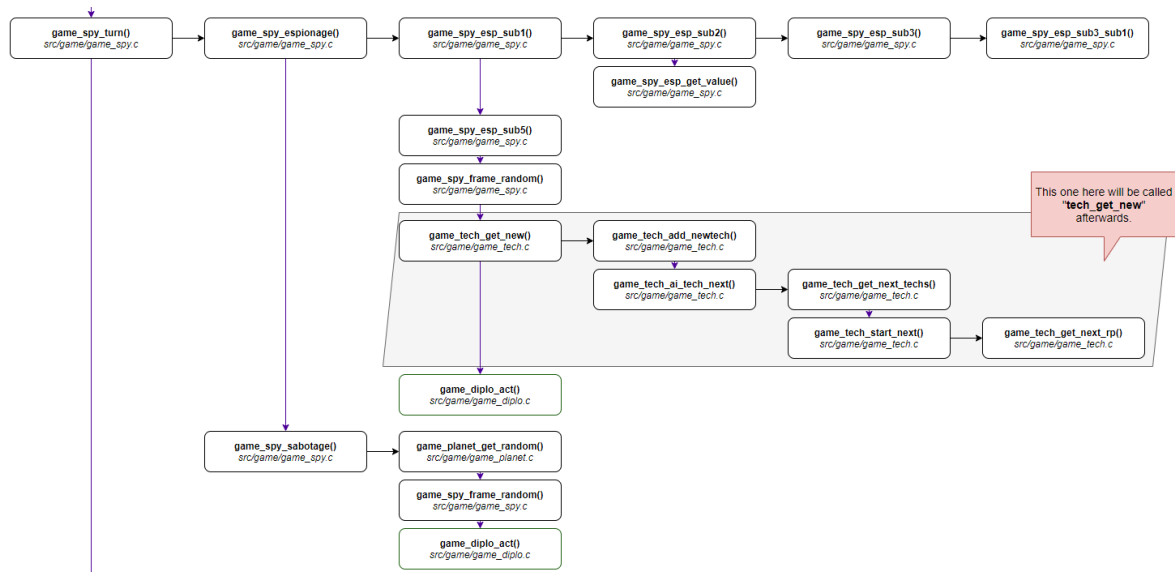
I can't say if such an approach would have worked better.

Maybe it would, but indirectly, with AI having more easy access to the specific tech and with the human player seeing the AI's priorities via spying.

Both work in the confines of the original system, though.

Spying

Here's how it looks ([full res](#)):



It's not the full picture.

I omitted the routines for human interaction with the system, the standalone function for building up spies and I didn't include p3 phase into it.

Time to pick up the slack!

One thing to note: the AI reassesses its espionage spending and priorities each turn starting from turn 31.

On a small map with five opponents that probably may be exploited. Usually it's a sensible choice, though.

Spending and priorities

MOO1 has both espionage and counterespionage.

Counterespionage is just spending, espionage is spending plus focusing your spies on some action: hiding, stealing tech or sabotaging.

The logic I omitted in p3 handles both spending and priorities.

A human can set up anything in-between 0 and 10% of total empire budget on spying against a particular empire and anything in-between 0 and 20% on preventing others from spying.

The AI operates roughly by the same rulebook.

It counts its spending in tenths (i.e. a 5.4% spending is 54 for an AI), but so do you, you just don't know it. Remember, no floating points.

The basic level of counterespionage level is $8 * \text{difficulty level}$, i.e. 0 for simple, 8 for easy and so on until 32 on impossible.

Then with a 50% chance it adds the total number of spies other players have sent to this AI's empire.

The AI's counterespionage level can't exceed 100 though, or, in human terms, 10%.

The espionage spending is 6-10 tenths or 0.6-1.0% for each target, decided randomly.

The espionage priorities were probably meant to be more elaborate, but that segment of the code (`game_ai_classic_turn_p3_sub1()`) is unusually bug-riddled.

I'll explain how it works in practice.

By default the AI just sets spies to hide.

If the AI is at war with the target empire, it always sets spies to sabotage.

With 100% probability if the AI is Darlok and with 50% probability if the AI is non-Darlok, the AI will consider espionage. The chance of doing nothing is $100\% + \text{level of relations with the target empire}$ (which ranges from -100 to 100).

But that's not all.

By default any AI's spying stance is "hide spies, spend nothing".

If an AI doesn't have an espionage or sabotage mission "in mind" against an empire, it won't spend a single BC on supporting the spies that are already there.

And while the AI will update the spying priorities and apply espionage spending against the human player each turn, against other AIs it will do it only 2 times out of 3.

Ok, we've got the targets, we've got the (very modest) budget.

How does the spying actually go?

Questioning authorities

Now I find myself in a pretty uncomfortable spot.

I do not see any advanced logic described in the official guide on pages 280-283 in the code, but I don't see it mentioned in Kilgore Trout's errata either.

Finally, I chose to believe that 1oom is a faithful recreation and these things weren't mentioned in the errata just because.

It's absolutely valid to think otherwise and doubt the faithfulness of 1oom in this part, it's just I chose the other option.

```
/* BUG v is uninitialized and overwritten anyway */
```

Contrary to what the otherwise excellent guide says, an AI in MOO1 (or, technically, 1oom) does not:

- spend more on spies if it's at war with someone
- have any behavior related to spy wars level 1, 2 and 3 (theat count, level escalation, shifting the percentage between espionage and sabotage, etc)
- reevaluate espionage spending after researching a tech
- have any trait-related objectives described in table 12-5
- send spies into hiding after them failing a security check
- spend 8-10% of budget on spying against a single race or at all
- add 1% of budget to the counterespionage fund against each spy sent in

In short, the AI doesn't even attempt to do any of these nice things.

Again, for me is unfathomable that Kilgore Trout could have missed such a trove of code during his disassembly process.

But why didn't he mention it in the errata?

I just don't know. I'll just describe what I see.

If that means I'll be repeating the official guide a lot, so be it.

1, Spying process

Converting BCs into spies

The spy in MOO1 costs from 25 to 125 BCs to produce.

It's 25 base cost plus twice the empire's level in computers.

For Darloks the spy is half the price.

But wait, that's the first spy.

The next one costs twice more.

And the one after it costs twice more again, so four times. And so on.

If you're waiting for me to add "for this empire" you'd be correct, but for 1oom.

MOO1 didn't reset the cost of spies between target empires due to a bug, so only the target empire with the lowest id got the intended number of spies and any other empire would be lucky to get one.

They eventually do get a spy, as all money left unused this turn will be saved for a spy build attempt next time, but once AIs start to spy on the human player (which always has id 0), they effectively drop any spying on each other in MOO1.

Infiltration

Ok, we have a non-zero number of spies at the target empire.
Each turn they try to infiltrate (or hide).

Two variables are in play here:

- evasion — spy computer level - target computer level
- detection — target computer level + 2 * counterespionage spending percent - spy computer level

There's a subtle nuance.

While Darloks enjoy a +20 bonus to checks when they're targets of espionage, when they themselves spy, they get a +30 bonus.

So if we imagine Darloks spying on Darloks with an equal computer tech level and no counterespionage, we'll get evasion 10 and detection -10.

Both chances can't be less than 0.

If the empire told its spies to hide, each hiding spy has a 15% + detection chance to be caught.

If they were told to do something, the stakes are higher. Each spy rolls the dice and:

- with 50% + detection chance the spy gets noticed and can't do anything
- with 30% + detection chance the spy gets caught
- with 1% + detection chance the spy gets caught and busts all other spies as well

Now all surviving non-hiding spies can go on the offensive.

With 15% + evasion chance they'll go on with their sabotage or tech stealing mission.
And with 0% + evasion chance they'll even be able to frame someone else.

Espionage and tech grouping

The AI in general divides all techs in the game into groups (listed in the table in appendix).
That applies to human player's spying as well.

There is a special group with 0 id which is considered essential and all techs in it are considered equal.

All other techs are non-equal and have some intrinsic value assigned to them (not to be confused with tech value we'll see used in diplomacy using mostly the same espionage routines!).

So, group 0 is things like hyperspace comms, high energy focus or advanced cloning. Can't compare them.

But, for example, group 11 is planetary shields. These techs can be compared.

These intrinsic values are 1, 2, 3, etc and are with some exceptions consecutive.

The spying empire will tally up all techs researched by the target empire and start to strike out the non-interesting ones.

It will strike out all techs it already has.

If it's a Silicoid empire, it will flatly mark reduced industrial waste, eco restoration and colonization techs as uninteresting.

And it will strike out techs that are worse than what it already has, as in that have lower intrinsic value.

But not everywhere, just in the following groups: improved robotic controls, reduced industrial waste, improved industrial, planetary shields, space scanners, fuel cells, ground troops weapons, ground troop armors, ground troop shields, terraforming, colonization, eco restoration.

That's a handful, but these are techs that are obviously obsoleting previous ones.

The empire may need a class III shield for medium ships even if it already has class VI.

Interestingly, engines and armor, usually considered as a strictly better upgrade, are omitted from this check.

Up to 6 techs are randomly selected from this preliminary list.

This is what I'll call the interesting list.

Each successfully infiltrated spy rolls the dice, from 1 to 100.

The maximum number rolled defines how good techs is the spying empire going to get.

The ceiling of spying is the maximum level the target empire has in any of the fields multiplied by that maximum number treated as percent.

So if I have two spies, one rolled 37, the other rolled 54 and the target empire's most advanced field is planetology at level 27, my ceiling is level 14 (not 15 due to the integer division).

With that ceiling in hand, the game goes on to eliminate techs from the interesting list in a very cruel manner.

If any tech in the list has a higher level than this ceiling, the first tech in the list is chopped off and the process repeats.

"Punishing the innocents and rewarding the uninvolved" at its finest.

There are interesting implications.

The more even the tech progress of the target empire is, the harder it is to spy at it.

If there are any techs left from this vivisection, the human player gets to choose the field and gets the first tech in the list in that field.

The AI just takes whatever is the first in the list overall.

If there were any spies caught, the human player faces repercussions, the debuff from 5 to 60 (random) in relations with the target empire.

If it's possible to frame someone, the human player may choose.

If the human player was the target and the AI can frame someone, it'll frame at random, friend, foe, ally, rival, whatever.

In spying between AIs the framing AI will frame the human player with 50% probability.

Freaking Darloks, how do they work?

Sabotage and revolts

The planet for sabotage is always chosen randomly.

There are three ways a sabotage can go for an AI (the human player gets to choose):

- incite a revolt — 25% probability against a human and 0% against a fellow AI
- destroy factories — 37.5% probability against a human and 50% against an AI
- destroy missile bases — 37.5% probability against a human and 50% against an AI

How much freedom and liberty will the sabotage inflict on the tyrannic regime of PLAYER_NAME depends on the number of spies infiltrated, the sabotaging empire weapons level and on pure luck.

The game rolls from 1 to 5 a certain amount of times and sums up the results.

That's the force of the sabotage.

Now that amount of rolls is $(\text{number of spies} * (\text{weapons tech level} + 9)) / 10$.

If I have two spies infiltrated and my weapons level is 13, I'll get 4 rolls and my sabotage force will be anything from 4 to 20.

Let's say it's 12.

What do I get for 12 sabotage?

- 12 factories destroyed

or

- $12 / 5 = 2$ missile bases destroyed

or

- $(12 / 2)\% = 6\%$ of the population (but no less than 1 pop) on the target planet joins rebels

Replace “12” with “sabotage force” and you’ll get the formulas.

While factories and missile bases are trivial, rebellion works a bit differently.

Things look normal, but once the number of rebels (cumulative, several empires can in theory incite a rebellion on a given planet) is more than the half of the population, the planet goes rebel, against all authorities.

It’s formally still owned by the host empire, but doesn’t do anything, doesn’t pay anything and requires... a population transfer for things to return to normal.

As Kilgore Trout correctly notes, the rebellion can’t spread despite the guide saying so.

However, if more than half of the empire’s colonies are rebelling, the emperor gets deposed. For an AI that means a new trait set will be picked, for a human player that means game over.

Interestingly, repercussions are milder, the diplomatic debuff in relations ranges from 21 to 40 for the human empire.

As usual, the AI frames just about anyone if it can frame.

Thoughts

I believe Steve Barcia had all this advanced logic in mind, but simply ran out of time to fully flesh out the p3 phase.

The espionage itself looks fairly developed, it’s even deeply intertwined with the byzantine structure of diplomacy and audiences.

What’s left rudimentary are the decisions on whom to spy, with what expenditures and why. Just like with the research decisions.

I’m quite saddened by this, because in my opinion if what the guide describes was implemented and indeed worked as intended (not a given), MOO1’s espionage would still be the best out of all 4X games released since.

Granted, it’s not a high bar to clear. Not many 4X games bother with espionage anyway.

Sometimes it’s made to check a box in the features list, sometimes whole DLCs are centered around it (like Endless Legend did and Stellaris plans at the time of writing). The latter usually turns out less playable than the former.

Espionage as it is present in 4X games is glorified gambling.

The dice roll and someone wins and someone loses with little to no influence from the players.

The only thing you can do is decide how much you want to bet.

I believe it’s caused by:

- the absence of small bets
- the different context in which spying operates

The only real treasure your spies can return with to you is the magic of friendship.
4X genre doesn't have that, so they return to you with the next best thing: technologies.

Usually there's no such thing as a partial technology or a clue to the technology as it requires a lot of design investments into the research process.

Usually you can't give a flying feather about what the AI thinks or where its fleets are because you're most likely geared for macro or for cheese.
Sure, in a setup with logistics and supply chains (like in idealized Star Ruler 2) you'd love to know where to strike for the opponent to crumble like a house of cards, but such a game doesn't exist.

Spies can't even make the diplomatic audiences more favorable to you because diplomacy in 4X is usually barebone.
We'll get to MOO1's diplomacy in a short while.

With no small bets on the table, the designers make it hard to win the large ones.
And so spying is notoriously unreliable and risky.

Ramp up the counterespionage, keep enough spies to know what tech your rivals have for trade and just kill them with the blasters they sold you.

Now, the context in which the player operates is terrain, colonies and fleets.
Spies do not register there. Spies do not exist there.

If a designer would've wanted spying more involved, with informed decisions and all that, it'd require a separate map, completely orthogonal to the usual one in terms of what information is displayed.
It would be a mini-game and everyone hates mini-games.

All these issues are not unsolvable problems.
What makes them real hard is the limited budget of the player's attention in which the solution must work.

There are stars to conquer, fleets to build, mighty forge-worlds to establish... and there are spies.
Which demand more attention than two billion sentient beings under your command and fifty black hole equipped ships combined.
Yeah, something in this equation seems more fun to me.

That said, espionage as described in the guide would've been fairly good.

I like how the chance that a caught spy will betray the whole ring balances the reward with risk.
That's completely unlike MOO2 where you just end up shuffling stacks of 63 spies between rival empires.

The spy wars level mechanic is of particular interest.

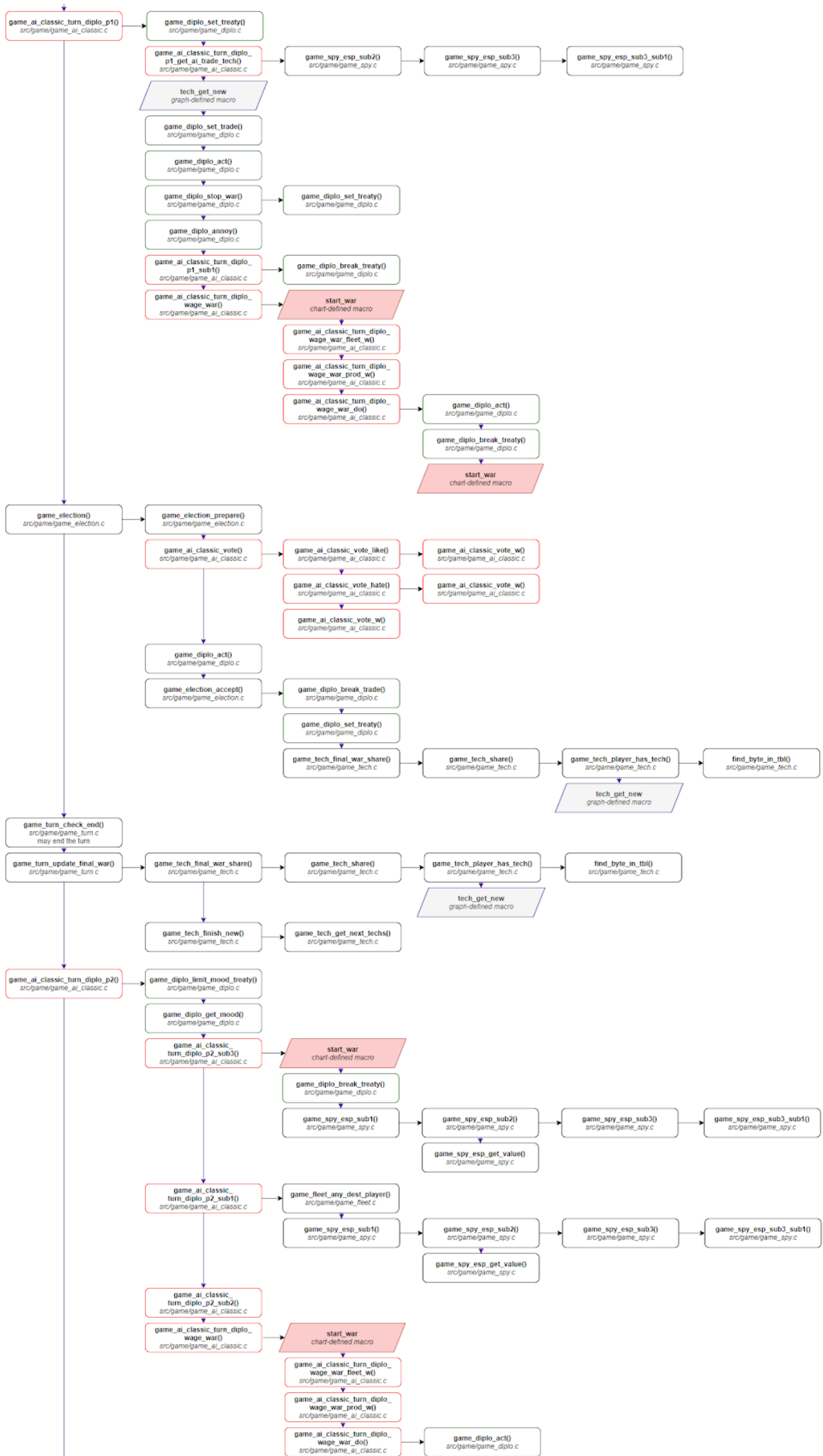
The more there are wars on the map, the more even uninvolved empires should be spending on spies, and at the top level the spending switches from 75/25 espionage/counter to 25/75 espionage/counter.

It would've made the whole spying subsystem more grounded, more reflective of what goes on on the map.

Alas, it looks like it never was built.

Diplomacy

Ok, here's how it looks ([full res](#)):



That's a lot:

- Als are dealing with Als (diplo_p1)
- galactic senate elections, if it's the election year
- if some human player defied the election results, the final war starts
- Als are interacting with the human player (diplo_p2)
- the human player is interacting with Als (audiences)

There's more, there's a trivial "update mood and diplomatic blunders" function at the start of the turn, more complex "diplo adjust" function called before diplo_p1 and "mood relaxation" at the end of the turn.

So I'll omit the final war stuff as it's fairly trivial. We're in for a ride anyway.

While the diplomacy is the only way for a human player to "humanize" the AI and feel that it's an equal opponent, so a lot of complexity is justified, the procedural programming style makes it a nightmare.

Frankly, if I didn't draw the map, I'd balk at describing this subsystem. It's a byzantine state machine.

My hat's off to Kilgore Trout for a lot of reasons, but if I had to pick one, it would be recreating this.

```
/* M001 does unused buggy count of planets ;  
   |   overwrites local variable at tbl[-1] (which is also unused) */
```

I apologize in advance for the length of some parts.

I omit and abstract what I can without losing the clarity, but it is a very, very procedural code.

Main concepts

Diplomatic act

The main primitive the game uses to change relations between empires is *game_diplo_act()* function.

It abstracts a lot of very interesting functionality from the diplomatic routines, so it's only fitting to start here.

Empires can do good things to each other and bad things to each other.

How much an act is pleasing is defined by its diplomatic value. More than zero, it's a good thing, less than zero, it's a bad thing.

However, this initial diplomatic value gets modified heavily before it's applied.

Good things done to empires with negative relations count twice as much as they're worth.

However, no single good deed should make you best friends, so the diplomatic value is capped by (10 - initial relations).

That means whatever you do, an empire will get up to 10 relations with you for this act.

Good things done to empires with positive relations count less and less after you surpass the 24 relations barrier.

In the bracket between 25 and 49 your good deed will count twice less, in 50 to 74, thrice less, etc.

Bad things done to empires with positive relations hurt twice more than they should. There's no backstop here.

Bad things done to empires with negative relations count less and less just like good things done to friends.

If you're at -24 to 0 range, this act will sour your relations by its nominal value, at -49 to -25 it'll hurt twice less, etc.

After this modification, some ethics traits come into play.

If the initiating empire is:

- honorable and does a bad thing, the value is multiplied by 2
- xenophobic and does a bad thing, the value is multiplied by 1.5
- xenophobic and does a good thing, the value is multiplied by 0.75

At this point the current state of the diplomatic value of the act is stored as a peace value. It'll be used later.

But we still have to modify the diplomatic value a bit more.

If the receiving side's race is human, they value a good thing twice more.

If it's already turn 101 and higher, the AIs get to cheat when dealing between themselves:

- the value of good things is multiplied by 2
- the impact of bad things is divided by 2 on average and hard and divided by 3 on impossible

Now, the existing level of relations between empires is added to the diplomatic value, the result gets clamped between -100 and 100 and we finally have the diplomatic value that'll be applied.

The relations between empires are set to this new diplomatic value.

If the empires are at war, the new relations can't be higher than -25.

If empires are allied, the new relations can't be lower than 65.

This new diplomatic value is added to the mood for treaties between these two empires, to the mood for trade treaties and to the mood for tech trade.

The mood for peace, well, it's complicated.

If that peace value we talked about before is negative, a quarter of it will be added to the mood for peace of the initiating empire and subtracted from the mood for peace of the receiving empire.

As it's negative, the initiator will want peace less and the recipient will want it more.

If that peace value is positive, it just gets added wholesale to the mood for peace of the receiving empire.

So, if you make something good to an empire you have bad relations with, the relations get better, but, as the final diplomatic value is still negative, moods for treaty, trade and tech take a big hit.

One last thing to note, the final diplomatic value from the last interaction is stored and is frequently used by an AI when dealing with the human player.

Primary moods of the AI

But what are these moods?

Oh, the AI in Master of Orion 1 has a lot of moods.

Out of ninety-something members in the structure that describes the state of the empire, about a half is dedicated to moods and opinions about everybody else.

“Do I want peace”, “do I trust them”, “what are our relations”, “what should I ask for in a tech trade”, etc.

Primary moods are:

- mood for a treaty, i.e. a non-aggression pact or alliance (ranges from -200 to 120)
- mood for a trade treaty
- mood for a tech trade
- mood for peace

Technically, each empire has a separate set of moods for every other player.

In practice, all primary moods except for mood for peace are the same between every pair of empires, as ensured by the diplomatic act function.

Each primary mood is bumped at the beginning of each turn up by 1-5 (random) if it's lower than 50.

If it's lower than 0, it gets another 1-5 (random) bump.

They get adjusted before the diplomacy rounds through a slightly more complicated round of checks.

For a non-aggression pact or a trade treaty, players get a diplomatic act with 1-3 (random) initial diplomatic power.

For an alliance, the initial diplomatic power of the act is 1-6.

A human player gets a diplomatic act with -10 power from an AI if the human player's fleet is stronger (in total hull weight). But it happens rarely.

It is gatekept by two checks, one has 5% success probability, and the other quickly drops from 50% when fleet weights are roughly equal to 0% when the human fleet is at least twice as powerful.

Each even turn there's a diplomatic act that with 25% chance slightly better the relations with allied AIs if the player (AI or human) has more productive (more than 100 BC after maintenance costs) planets than "allowed" on this map size (3, 6, 9 and 12).

And at the end of the turn after everything is said and done, all the primary moods are bumped by 10 if they are lower than 100.

Secondary moods of the AI

Primary moods do not cover all the behavior modifying counters in the MOO1 AI. However, I'll omit those that are rarely used.

What's left is what I call secondary moods: relations, trust and hatred.

Relations are what you see in the diplomatic UI. They range from -100 to 100. Initial relations are:

	Human	Mrrshan	Silicoid	Sakkra	Psilon	Alkari	Klackon	Bulrathi	Meklar	Darlok
Human	0	12	12	12	12	12	12	12	12	12
Mrrshan	12	0	0	-24	0	-36	-12	-12	0	-12
Silicoid	12	0	0	0	0	0	-12	0	12	-12
Sakkra	12	-24	0	0	0	-12	-12	0	-12	-12
Psilon	12	0	0	0	0	0	0	0	0	-12
Alkari	12	-36	0	-12	0	0	-12	0	0	-12
Klackon	12	-12	-12	-12	0	-12	0	0	0	-12
Bulrathi	12	-12	0	0	0	0	0	0	0	-12
Meklar	12	0	12	-12	0	0	0	0	0	-12
Darlok	12	-12	-12	-12	-12	-12	-12	-12	-12	12

Technically, there are two relation meters, the main one and the backup one that gets impacted slightly for breaking an alliance or declaring a war.

This little-changing backup is used to reset the relations with the empire after it suffers a coup. "Everything is forgotten except for that one time!"

Usually though the AI racks up enough wars so that the new government will wage war with the same opponents as before almost immediately.

Trust is a strange parameter.

It gets used often, yet there is only one way to lose it and only one way to gain it.

It ranges between -100 and 30.

The trust is lost only by breaking a non-aggression pact (-10 or -20 if honorable) or by breaking an alliance (-20 or -40 if honorable).

The trust is regained only by gifting tech (2-10 at random).

Hatred is something that AIs feel only towards the human player. It ranges from 0 to 3. Level 1 corresponds to warning diplomatic stances (“don’t do that”) or threats to break a treaty.

Level 2 corresponds to breaking a treaty.

Level 3 corresponds to a chance of starting a war.

Hatred grows each turn if the level of diplomatic rapport plus the AI’s mood for treaty with the player plus 100 is less than a random number between 1 and 100.

Hatred is reset to 0 if an AI wants to have a deal with the human player.

If the conditions for attempting a deal are met and hatred is more than 0, it’s reset. Next time, the AI will propose a deal.

Hatred is reset to 1 if the war is started.

Diplomatic rapport and traits

But what’s that diplomatic rapport I talked about?

It’s my short-hand for an often used sum of the:

- trust the initiating empire has in the responder
- relations level between them
- value of the responder’s ethics trait.

In diplomacy that involves the human player 50 bonus is added to the rapport if the responding party plays for the human race.

Between themselves, AIs do not apply it.

Traits’ diplomatic values equal to:

Xenophobic	Ruthless	Aggressive	Erratic	Honorable	Pacifistic
-50	-30	-20	0	10	20

The human player almost always starts as a xenophobic diplomat. That’s fair.

That happens unless the human player plays for the human race, then the effective checks go as for an erratic diplomat. Also fair.

AI’s diplomatic process

Deals between the AIs

First things first, the AIs will strike deals between themselves. No humans allowed.

Each pair of the AIs has from 1/15 to 1/7 chance (from the easiest to the hardest difficulty) of having an “audience” each turn.

Due to how the loop is arranged, each pair is checked twice, so the chance is actually from 29/225 to 13/49.

First of all the level of diplomatic rapport is determined and a random number from 1 to 100 is added to it.

This value is further enhanced by existing:

- non-aggression pact or alliance (plus 10 or plus 20)
- trade agreement (plus 10)

Now the AIs go on and try to choose a treaty to sign.

They can do one of the following:

- if the rapport value plus the responding empire's mood for treaty is more than 150 and these AIs are not in an alliance, they'll strike an alliance
- if the rapport value plus the responding empire's mood for treaty is more than 150 and these AIs are not in a non-aggression pact, they'll strike a NAP

Due to the coding oversight the AIs that are in very good relations will constantly oscillate between signing an alliance, overwriting it with a NAP, overwriting it with an alliance, etc.

As AIs are made to strike just one deal during an audience, these BFFs rarely get to:

- if the rapport value plus the responding empire's mood for tech trade is more than 80, the AIs will exchange random techs they have (but not the best ones, the techs are at least 1-3 levels lower than the state of the art for the empire)
- if there's no tech trade, but the rapport value is higher than 70, the AIs will sign a trade treaty amounting to 1/4th of the weaker empire total production (but no more than 32000 BC)

Now the original Master of Orion replaces the value of the rapport with the index of the last message in the audience, which can be anything from -1 to 13, but usually in -1 to 7 range. Yes, you read that correctly.

And with that modified rapport the AIs will try to stop the war between them.

If the rapport plus the responder's mood for peace are greater than 100, the empires stop fighting.

That's why original AIs can make peace with a xenophobic empire roughly with the same effort as with pacifists.

Whether the AIs succeeded in signing something or not, with 95% probability their treaty, trade and tech trade moods will be slightly improved by passing a diplomatic act with the force of 1-5 (random).

But that has little sense as the empires inevitably diplomatically annoy each other, and all their moods (trade, treaty, tech and peace) are decreased by 30.

And then AIs will try to start a war with each other. Again, inevitably.

Starting wars between AIs

An erratic AI has 0/300 to 4/300 chance (from the easiest to the hardest difficulty) to start a war for literally no reason.

That means that on the hardest difficulty with 5 AIs it has about 49% chance to start a war in 50 audiences, and that means about 48 turns once everybody knows each other on a long enough run.

A non-honorable militarist or expansionist with 1/20 chance will consider war if its fleet is much stronger than the victim's.

Any AI will consider war with 1/20 chance if its production is much stronger than the victim's.

How does that "much stronger" work? Through a lot of headache.

For fleets it's weighing up all your fleets by hull weight, weighing up the potential victim's fleets by hull weight, dividing your weight by the victim's and subtracting 1.

For production it's the same, just for the total empire production in BC.

Now the fun part starts. Multiply that by 100 and clamp between -300 and 300.

Then, use a random value between this result and 0

Yup. That's not all.

By the logic of the code the victim empire "calls" for a war with itself.

If that random value we just got plus the rapport from the side of the victim empire (i.e. relations with the aggressor plus trust in the aggressor plus the ethics trait of the aggressor) is less than -150, the war is declared.

There's also a special case for the human player.

If the difficulty is average or harder, and the human player has less than 2, 3 or 4 wars (average, hard or impossible) going on, any AI that has relations lower than -30 with the human has a chance to declare war.

The chance is that negative relation, but in percent. If it's -50, the chance is 50%.

Senate elections voting

Ok, there are two candidates in the election (each 25 years after a certain date after $\frac{2}{3}$ of planets on the map are colonized).

The first candidate is the most populous empire, the second candidate is the second most.

How does an AI decide whom to vote for?

Let's deal with the trivial matters first.

An AI will always vote for itself.

If the AI is in alliance with the first candidate, it'll vote for the first one.

If the AI is in alliance with the second candidate, but not with the first, it'll vote for the second one.

If the AI is in war with the first candidate, it'll vote for the second one, no matter what.

If the AI is in war with the second candidate, but not the first, it'll vote for the first.

Ok, no war, no alliance, no representation. Now how does the AI choose?

If the AI has neutral or better relations with the first candidate, it checks if it likes the first candidate. If it does, it votes for the first candidate.

If the AI has negative relations with the first candidate, it checks if it hates it enough to vote for the second candidate. If it does, it votes for the second candidate.

If the AI is still undecided, it evaluates the second candidate.

Again, if relations are neutral or better, it checks if it likes the second candidate, and if it does, it votes.

If relations with the second candidate are negative, it performs a slightly more strict hate check to vote for the first candidate.

If the AI is still undecided, it abstains.

The only thing left to explain is like and hate checks.

The AI likes the candidate with the probability equalling to $\frac{1}{4}$ of the relations.

If the candidate owns Orion or is human (race, not player), it gets an additional 20%.

If this AI voted for this candidate in previous elections, it gets an additional 40%.

So, the AI likes a human empire with 44 relations with $44/4 + 20 = 31\%$ probability.

The hate check is the like check, but with $\frac{1}{4}$ (or $\frac{1}{5}$ in the second hate check) of negative relations with the other candidate as a bonus.

So -20 relations with the other candidate transform into +5% or +4% for this candidate.

AI's deals with the human player

Most of the checks here are "should I start a war", "can I start a war", "pretty please, a little war, okay?"

When the AI is done with the greetings, next time it'll inevitably look for an option to start a war. Or stop a war. They're in the same subroutine.

If the rapport between the player and the AI or the minimal out of all primary moods between them is -100 or less, well, time for war.

If the last interaction was bad and if there's some hatred towards the player from the AI, oh, let's check if we can go to war. If it was bad and there's no hatred, the AI goes straight to the war.

If there is a war and the AI's mood for peace is higher than 30 plus a random number from 1 to 100, well, time to sweeten the deal and ask for peace.

Only if there's no war and no bad blood the AI gets to the final 1 chance out of 4 check to try to make a deal with the human player... with the probability of the diplomatic value of the last interaction

There's also a subroutine that ensures war declarations in an alliance, but it's trivial.

Starting/ending a war

The AI will flat out start the war with the human player if relations are -95 or less.

If the relations are somewhat better it will go through the hate routine, described above. Once the hatred level is 3, the AI will start rolling dice for a war.

The chance to start it usually is 25% plus the absolute value of the last interaction (assuming it was negative). For humans it's 0% plus the absolute value of the last interaction.

However, if there's already war, and rapport plus the AI's mood for peace plus a random number from 1 to 100 is 50 or higher, then with 1 in 8 chance the AI will ask for peace.

It'll sweeten the deal if the human player's total empire production is high enough. There's 15% probability of sweetening if the human's empire is twice less powerful economically than the AI's, 30% if equal, 60% if twice more powerful, 90% if trice, etc.

With 25% probability the AI will offer a random tech as a sweetener. Not the best tech, though. It'll search from the very basic to two levels below its state of the art.

Otherwise, it'll offer money.

Half, one third or one quarter (random) from what the human player has plus the number of turns since the game started. This sum is "rounded" to 25s.

Making a treaty

Even if all conditions to start the deals making subroutine were met, the AI can still have second thoughts.

It won't start the process if any of the player's fleet is en-route to its planets. And even after with a fifty-fifty chance it'll decide to just not to think about the deals this turn.

Then the AI goes down through a long list of all possible deals. If a deal meets the AI's requirements, it won't go any further. Only one deal can happen per turn.

The rapport is heavily used here and if the initiating AI plays as humans, it gets that +50 diplomatic bonus.

So:

- 1) **non-aggression pact:** proposed if there's no NAP or alliance, the relations are higher than 15 and rapport plus AI's mood for treaty plus a random number from 1 to 100 is more than 50;
- 2) **alliance:** proposed if there's only a NAP, the relations are higher than 50 and rapport plus AI's mood for treaty plus a random number from 1 to 100 is more than 100;
- 3) **trade treaty:**
 - a) considered if rapport plus AI's mood for trade treaty plus a random number from 1 to 150 is more than 50;
 - b) the AI randomly selects a trade value between 0 and quarter of a weaker empire's production in 25BC increments;
 - c) if this value is higher than the current trade treaty, a new one is proposed;
- 4) **bounty for an attack:**
 - a) proposed with 10% probability if there's a third player the AI is in contact with and the human player is at war with;
 - b) offers tech as in sweetening for peace, if no tech is found, offers a minuscule amount of money;
 - c) thoroughly bugged, bounty is never paid off in the original, the AI should offer human to attack empires the AI is at war with, not vice-versa;
- 5) **tech trade:**
 - a) considered if rapport plus AI's mood for trade treaty plus a random number from 1 to 100 is more than 25;
 - b) the AI will spy on the human player and pick the very first tech it finds;
 - c) the AI will spy on itself "as" the human player and find a number of techs that in AI's eyes don't cost more than twice than the tech the AI wants;
 - d) the AI proposes a tech trade;
- 6) **pity NAP?** : if the relations are bad and the AI is more powerful economically than the human player, the AI will offer a non-aggression pact; probably the players are mixed here again.

When everything is said and done, if the human player plays as humans, the AI may gift something.

The checks and rewards in place are very similar to the case when the AI wants to sweeten the peace deal:

- the primary check is the same 1 in 8 chance
- the secondary check is very close, just the positions are inverted: there's 15% chance if AI's empire is twice less powerful economically as the players, 30% if they're equal, 60% if the AI is twice as powerful, etc
- very similar choice of rewards:
 - 25% chance of a random tech
 - half, one third or one quarter (random) from what the human player has plus the number of turns, "rounded" to 25s.

It's nice to be a human in Master of Orion 1!

Human player's deals with the AI

We're at the final stretch.

Most of the concepts are now familiar to you, there's just that last bit of complexity.

I'll omit some parts that are in effect identical to what was already described, for example, tech trade. I'll frankly skip over the actual audiences workflow, you just need to know that every action annoys the AI (i.e. sours all primary moods) by 10.

Mood check

Everything you as a player propose at an audience passes a mood check.

It is a special function that handles all four primary moods.

First of all, a familiar rapport is calculated in a familiar way.

It's AI's trust in you plus relations plus 50 if AI plays as humans minus 50 because of the human player's xenophobic trait.

As usual, a random number from 1 to 100 is added to it.

As not usual, if you're allied with that AI, a +40 bonus is added to the rapport.

Now, the mood that is checked is added to the rapport and the challenge value the check has to beat is subtracted.

We've got some final number.

As opposed to AI-initiated deals that just don't go further if a check is failed, this function has several severities of failed check.

If the final number is zero or higher, that's fine, we passed the check.

If it's from -50 to -1, well, that's a problem, the AI will ask for money in return.

If it's from -75 to -51, that's a serious problem, the AI will ask for tech in return.

And if it's less than -75, that's a no-go, the AI won't agree to the proposition.

Oh, and by the way, the primary moods get soured, whether the check passed or no.

All moods are soured by 10 and in addition:

- if mood for treaty was checked, it's decreased by 21-50
- if mood for trade treaty was checked, it's decreased by 21-50
- if mood for peace was checked, it's decreased by 51-100
- if mood for tech trade was checked, it's decreased by 21-70

Diplomatic options

Now I'm afraid I didn't find any evidence for the time-honored strategy of making a minimum volume trade agreement with an AI and then upping it by 25BC to prevent the AI from attacking you.

That may be a placebo, after all.

Ok, so what are the challenge values to beat?

- non-aggression pact: you need to have relations better than 10 and the treaty mood check should beat the value of 50
- alliance: you need to have relations better than 50 and the treaty mood check should beat the value of 125
- peace treaty: the treaty mood check should beat the value of 60 (peace mood check in "Classic+")
- declare war on third party: the treaty mood check should beat the value of 150 + responding empire's relations with this third party (should be 250 in case of a NAP or alliance, but alas)
- break alliance with the third party: you need to have 25 or better relations and the treaty mood check should beat 175 + responding empire's relations with this third party
- trade treaty: the trade treaty mood check should beat the value of 50

Non-diplomatic options

You can also demand things from an empire.

The force of your demand is aggregated from several parameters.

It's half the mood for treaty plus double the weight of your ethics trait (in essence, -100) plus a random number between 1 and 200, plus... the ratio between their production and yours multiplied by 100?

Yeah, the smaller you are, the better you can demand things, apparently.

Of course a demand sours relations by 1-15 points, it puts the mood for treaty at whopping -120 (so you can't demand for several things in a row), but here's what it gets you.

If the force of your demand is less than 170, with 51-65% probability the AI will declare war on you.

If the force is 170 or higher, the game will hide your spies. It's a bug, of course, the intended action is making the intimidated empire cease spying.

If the force is between 200 and 274, you'll shake down some money from the AI.

If the force is 275 or higher, you'll get a random tech the AI has and you don't. If it exists.

Thoughts

Diplomacy turned out to be the largest and the hardest part to describe.

If you think now that Master of Orion 1 is a diplomatic state machine with some stars and ships bolted onto it, you're not alone.

It's fascinating to think how much effort went into creating and tuning this state machine. And for a good cause.

When I think about times I played Master of Orion 1, I rarely recall the map. Usually the image of audiences springs to my mind.

I believe the diplomatic illusion is what MOO1 did masterfully and what is sorely absent from modern 4X games, even the best ones, like Civilization V or Stellaris.

Well, there's Alpha Centauri, but it can drink in most jurisdictions now.

I did a fairly mediocre job of describing the diplomatic AI, because it's frankly too complex to convey this easily.

What I aimed for is to create an image, a source for imagination: these are the mechanisms, these are the relative values and here is a somewhat close description of how it works this way.

I'd like to underscore the idea of various moods.

Wouldn't it be easier to just rely on the relations meter like everyone else since did? Yes, of course.

Would it be better? I don't think so.

The moods may be a bit overcomplicated for what they deliver here, but their waxing and waning adds the element of play into the system.

Essentially there's a diplomatic cost to each and every action in the audience and it's up to you to use them in the best way.

I think making the red 'X' turn into a green checkbox through bought favors is not really a game.

Trying to balance techs, money, ships and planets in some kind of a primitive debit/credit tally is not a game too.

It's a service element of a game, but not a game.

Now knowing that going too far or asking for a fruitless audience may easily spiral the already not so good relations into a war, that's a different story.

Add a bit more manageable risk and tech trades won't be that overpowered anymore.

Overall I think that Master of Orion 1 was hindered by the language.

I like C and it's a Turing-full language, so you technically can do anything in it.

However, the juggling of various states the state machine can be in gets... very complicated really fast.

This diplomatic AI is, despite its bugs and shortcomings, a marvel of gritty engineering. More importantly, it was made with the eyes on the prize: how to entertain the player.

But we've got much better tools now. May as well put them into use.

Afterword/soapbox

I'd like to express my eternal gratitude to Kilgore Trout, whoever he is, wherever he is, whenever he is. May his novels finally start selling well.

Whenever I felt the road ahead of me was hard, I reminded myself that he travelled a path ten times harder.

Somewhere along the way, his dedication to preserving MOO1's behavior rubbed off on me.

What I initially thought to be a cursory exploration of a venerable game transformed into a task of design preservation.

I hope that I did a half-decent job.

In no particular order, I want to thank Battlemode, Jon Bellini (Alpha One), Jeff Gregg (Dominus Galaxia) and Ray Fowler (Remnants of the Precursors) for reading, commenting, encouraging and, most importantly, making me rewrite previously incomprehensible parts. I don't know Steve Barcia, but if anyone reading this knows him, tell him I said "Hi."

That said, I built myself a pretty good soapbox. So I'll stand on it and preach.

Paradigms

I believe that the field of antagonistic 4X AIs is well-explored already.

What they do is simulate a multiplayer deathmatch, because, let's face it, while everyone says they want to play a multiplayer 4X, hardly anyone shows up in the lobby.

At worst (or at release) the AI behaves like your usual FPS bots, at best (or modded) they are like competent human players.

StarNet AI mod for Stellaris exemplifies this approach best at the moment.

But the challenge, I believe, is just a single axis.

One good man in black pants said: "I'm not good at video games. I just enjoy playing them."

That's the majority of players in any game.

They aren't playing well, they just enjoy the game.

And that means the game's doing its work as it should. It should entertain.

That also means that the challenge is tuned for the game to be entertaining and then somehow scaled to get the green checkbox too from the more experienced crowd (low single-digit percent of sales and a significant amount of the forum posts).

That's the problem everyone deals with.

I'm not saying the antagonistic AI approach is a dead-end.

Absolutely not.

Games sell, there's something on the market for most players, the solutions for scaling difficulty are a-plenty and with good enough sales you may have modders that will pick up the slack.

The thing is, that you're entering into a deathmatch too, battling for the time of the player with everybody else in the genre.

Better bring a lot of money, you'll need it.

The only way out of this box, I believe, is to rethink the relations of the player with AIs.

Is a design in which the player will be interested in long-term, mutually beneficial relations with some AIs possible?

Can a player be motivated to vote for an AI to get some position of authority in-game?

Can alliances be simultaneously not of temporary convenience, but also not devolve into grinding levels for goodies?

I believe so. Plenty of designs are possible, even.

Things will change, though.

Hard to make a non-antagonistic AI when all empires are essentially land-locked and bump into each other's borders.

The Last Federation goes half the way there in regards to optional antagonism and way overboard in most others.

But I believe it can be done.

More than that, I believe that one of such novel designs can indeed be that coveted MOO2 successor.

I mean, whole teams have searched for it for twenty five years and found nothing in the antagonistic AI field.

Sure it makes sense to search for lost keys under a streetlamp, but maybe it's time to light another one?

Optimizations

I firmly believe that AI should be optimized for speed. Period.

Nothing you put into the design or animations is worth 15-20 seconds of the player's life each turn, let alone 30-40.

Part of the magic of MOO1 and MOO2 that they are blazing fast. On today's computers, of course.

But that's the only way we perceive them now.

Your game will be blazing fast on computers of the next decade. Maybe.

Still of little solace for you or your players today.

Long turn times (or long tick times in real-time) break the context the player was in.

Unnecessary camera transitions to yet another boring bombing only add insult to injury.

Find a way to show the player things worth caring about in a convenient fashion.

Same applies to tactical battles.

Sure, everyone wants for their tactical battles to be compelling, interesting, etc and looks up to tactical games.

But in a 4X tactical battles work in a larger context. Don't drag the player long enough to forget what the fight was all about.

In X-Com or Battletech the tactical battle is meant to take ~80% of the player's time. In a 4X it's just one thing out of the many. Don't make it a mini-game.

Multithreading is a life-saver here.
Also it puts quite a heavy load on the architectural design of your game.
You've got to plan for it from the beginning, and you frankly should.

Multithreading is a test of your integrity here.
It's easy to assume everyone has "at least" six cores and twelve threads in a CPU, so yeah, "works for me, no need to optimize further".
Sixty percent (at the time of writing) of Steam users will disagree.

What do you want more, for them to pay to upgrade or for them to pay for your game they happily didn't have to upgrade for?
I assume the latter if you're not in the hardware business.

There's also a very appealing idea of making AIs calculate their turns in the background while the human player thinks and drinks tea and then "just" recalculate what was invalidated by previous players.
Arcen Games wanted to do this when they were still making Stars Beyond Reach.

Technically that can be done through decision trees, but it probably will lay heavy shackles on the design.
May be very appealing for the programmers who want to make a game though.

But it's an even harsher test of integrity.
Such a decision allows to sweep things under the rug even further than multithreading.
Also, it raises an interesting question of how nice you should be to other processes that run concurrently to your game (and there will be hundreds of them).

Still, whatever you do, make it fast on your minimal system requirements and blazing fast on your recommended ones.
It's worth sacrificing everything for it, except the entertainment value.

Features

Games are designed for human players.
Ideas, features, mechanics, all that is there to entertain the player.

However, if the AI development is pushed to the end of the project, the question stands: how exactly should the AI handle all these features?
The answer is "Uh..."

Unfortunately, understanding how the game plays takes time.
A lot of time.

At least months and sometimes years until good players somewhat codify the approaches and modders, armed with these approaches, can make the AI competent.

Maybe reinforced learning will get convenient enough and affordable enough to QA the design, as in find loopholes that allow to stack modifiers, freeze the AI and all that. It'll be nice when it happens, but that future isn't here yet.

With tools and options we have nowadays it probably can be somewhat mitigated by always putting the AI first at the preproduction step.

"Here's the feature A, here's how the AI will use it and here's what's in for the player."

Sounds daunting.

A lot of good features will be put on a backburner and then forgotten because nobody figured out how to make an AI use it.

However, I personally can't call a mechanic fully implemented if it requires an external input (player's brain) to perform correctly. Partially implemented at best.

Yup, that goes contrary to current trends in game development. Can't blueprint this stuff. Nobody said it'll be easy.

However if you want your AI to be competent at release, your only other choice is to hope that you'll have enough clout to freeze all features that can impact the AI behavior for at least two-three months before the release. Good luck with that.

Conclusion

Make it fast, consider non-antagonistic AIs, put the AI usage first in feature design if you want it to be competent.

That's it.

The power of the ancient design is now yours to command.

Have fun.

Appendix

Homeworlds assignment

The game really doesn't bother that much when assigning homeworlds.

It just tries to assign them randomly and then checks if there is a colonizable (minimal or better) planet in the initial fuel range and is the distance to all other homeworlds larger than the minimal value.

Then, if the assignment went ok, for each homeworld:

- a) the chosen star's color is set to yellow and the planet is converted to terran with max 100 population;

- b) the planets are assigned 40-50 population, 30 factories and 50-55 initial production (depends on difficulty);
- c) the planets are renamed to the corresponding homeworld names;

But how does the random assignment of home planets even work if they are selected randomly and only 200 tries to place them are made?

Well, the checks on the minimal distance to other homeworlds and on existence of at least a minimal habitability planet in the starting game flight range are made only for the first two players.

These are the human player and the lucky AI that happened to have number 1 (human player is always number 0).

Why? Let's see why, we have the code.

Let's test the smallest and the biggest map sizes with the maximum number of opponents with and without that "only two first players" check.

We just need to add a couple of lines and then just test.

```
log_message("%i resets were made for %i players cared for out of %i on %i map size\n",  
            reset_number,players_cared_about_num,PLAYER_NUM,g->galaxy_size);
```

If this was a video, you'd now see a montage of test runs. Luckily, it isn't, so here are the cold hard numbers.

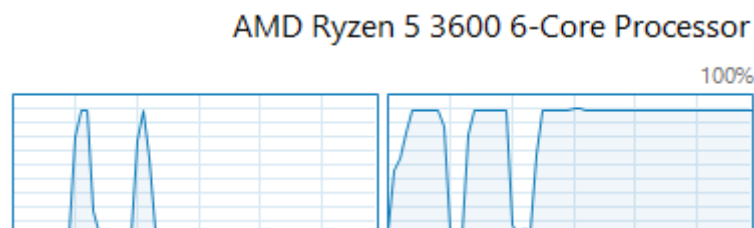
Not stats as I ran just about 15-20 tests for each combination, not nearly close to any statistically significant samples. It'll do for this case.

By default, with 2 players cared about from 6, we get the following number of resets:

	Small	Medium	Large	Huge
Average	6.92	1.82	0.11	0
Median	7	2	0	0
Min	0	0	0	0
Max	21	4	1	0

Nice. Let's care about all 6 players.

One small code change, recompile and let's start from the small map...



After about five minutes of waiting I gave up.

Medium maps take “just” about ten seconds to start on this CPU though.
So, here’s how the resets number looks for all 6 players taken care of:

	Small	Medium	Large	Huge
Average	DNF	114,4	56,6	2,5
Median	DNF	104	37	2
Min	DNF	57	1	0
Max	DNF	219	158	6

A lot of valid observations can be made from this exercise.

For example, it’s probably ok to ask the game to check for (2+galaxy_size) players’ well-being, so it’ll range from 2 on small to 5 on huge.

It’s astute to comment that placing the homeworlds first and then generating the map around them is prudent (and Remnants of the Precursors does exactly that).

But the most important observation I think, is that it’s not that important.
Really, more than 25 years passed and still nobody gives a damn about such a quick and lazy shortcut.

If you struggle with the fair placement of players, just make the AI cope with it and go on with your game.

Let’s find out how MOO1 made AIs to cope.

Initially and if there are planets nearby, MOO1’s AI sets planetary spending to 10% on ships and 90% on industry (don’t worry, *game_turn_process()* will call eco adjustment).
Computers, construction, propulsion and weapons are set to 20% research spending, while force fields are set to 15% and planetology to 40%. That’s 135% in total, don’t worry.

But if there is no colonizable planet in-range, the empire shifts into a different mode.

Industry is set to 60%, eco is set to 10%, research is set to 40% (110% total) and every single research point is pushed into propulsion to get a larger flight range as soon as possible.

The p3 phase countdown in this case is set not to 15-20 (as usual), but to 25-30 to ensure the AI doesn’t stray from the “I need better range now!” course.

The sum of sliders for production will be reset to 100% once the p3 phase runs for the first time.

The sum of sliders for research spending will be reset to 105% almost instantly, however, in 1-5 turns after the start of the game.

As was mentioned in the p3 phase overview, it’ll be mitigated: propulsion gets 75% of research before year 30 with 50% probability each time research priorities are reassessed.

It looks like a hack that was made earlier in the development was overridden by the spending adjustment system afterwards.
Still works somehow.

Traits assignment

An AI in MOO1 has two traits.

They were mentioned a lot of times in this essay, so I'll mention them again:

- an ethics trait — xenophobic, ruthless, aggressive, erratic, honorable or pacifistic
- a behavior trait — diplomat, militarist, expansionist, technologist, industrialist or ecologist

You've definitely seen them in the empire description in the diplomacy screen if you played the game.



The traits are assigned pseudorandomly.

Each AI player throws the same d10 or d12 die, but the outcomes are read from a race-specific entry in *game_num_tbl_trait1* or *game_num_tbl_trait2*.

Here are the probabilities (the official strategy guide gets them almost right):

	Xenophobic	Ruthless	Aggressive	Erratic	Honorable	Pacifistic
Human				10%	70%	20%
Mrrshan	10%	70%	20%			
Silicoid	70%		20%	10%		
Sakkra		10%	70%	20%		
Psilon				20%	10%	70%
Alkari				10%	70%	20%
Klackon	70%	10%	20%			
Bulrathi		10%	70%	20%		
Meklar	10%		20%	70%		
Darlok	10%	20%	70%			

	Diplomat	Militarist	Expansionist	Technologist	Industrialist	Ecologist
Human	41,7%	8,3%	8,3%	16,7%	16,7%	8,3%
Mrrshan	8,3%	33,3%	16,7%	25,0%	8,3%	8,3%
Silicoid	8,3%	8,3%	33,3%	25,0%	16,7%	8,3%

Sakkra	8,3%	8,3%	41,7%	16,7%	8,3%	16,7%
Psilon	16,7%	8,3%	8,3%	41,7%	16,7%	8,3%
Alkari	8,3%	25,0%	33,3%	16,7%	8,3%	8,3%
Klackon	8,3%	16,7%	16,7%	16,7%	8,3%	33,3%
Bulrathi	8,3%	N/A	16,7%	25,0%	8,3%	41,7%
Meklar	8,3%	8,3%	16,7%	25,0%	33,3%	8,3%
Darlok	25,0%	8,3%	8,3%	33,3%	16,7%	8,3%

Note how elegant the system of two overlapping pseudorandom choices is.

It allows you to design a three (like in this case) or a four-dimensional system using just two very readable tables.

Readability leads to controllability, you can easily see the gaps in your system and decide do you want them or not.

It's especially important at the preproduction stage. Let me perform a little skit.

Imagine I'm doing a trading game and so I need goods.

Imagine I chose the average cost and unit volume as axes (plural for axis as the Internet led me to believe) for table 1.

This covers basic things like sacks of potatoes, expensive trinkets, tons of sand and assembled machinery. Both you and I already know what kinds of ships a player may need for them. That gives me a basic and serviceable table that would work well if the main point of the game was something Privateer-like with pew-pew and all that.

I want a serious game about cargo, so I need more depth.
I need table 2.

Imagine I chose average margin and sensitivity as axes (now I feel like an orc chieftain from HOMM3) for table 2.

Average margin already imposes some restrictions on the game design.
To make it work, I need to ensure you can sell your goods only to designated (probably even willing) buyers.

And that makes sense, as you can't buy two kilos of apples at a supermarket and sell them at the kiosk down the street for 20 cents profit on each apple.

Sensitivity gives another dimension to the player's ship customization, as you can't transport champagne the way you transport sand.

But now I start filling the gaps in table 2 and I have to decide what I want to do with low margin, high sensitivity goods.

Should I leave them as joke goods? Or is it something that you can eek a profit from if you aren't allowed into a normal market by some customs or trade cartels? Interesting.

And high margin, low sensitivity goods, why doesn't everyone carry them already and drive the profits into the ground?

Novelty goods? Maybe. Investing into something new nobody knows how to sell yet is a good risk/reward scenario, but without additional information for the player to make an educated choice it's just glorified gambling.

Customs and border tariffs? Maybe. Different states and everything.
Don't want to go into a binary illegal/legal bit that leads just to the "choice" of installing a smuggler's hold.

Oh, it all can be combined into different interstellar states (I decided it's a space trading game somewhere along the way) with different economies, currencies and very little information transfer speed!

Just like medieval trading!
And the player gets better in the game just by traveling and paying attention!
New game+ is just restarting with all that knowledge! This changes everything!

Somebody, clip that! And give me the Games Workshop number, I need a WH40K license!
They give them out like candy, but the IP still has value somehow.

Hello, GW? What do you think about a trading game? Rogue trader? It's not a rogue-like!..
What combat? Servo-skulls? What do you mean by heresy?..
I warn you, I'm a citizen of Cargonia and I'm not afraid to use my axes!

Now, this silly skit was all about I, I and even more I.

That's because even if you do want to make a trading game now, you're likely to choose different axes and make different decisions looking at your own gaps in your own tables for your own game. The approach still stands.

It doesn't scale beyond four independent parameters, though, but rarely any system in the game design space requires more.

Handling higher dimensions easily is reserved for trans-dimensional species only and this essay doesn't cover MOO2.

Pi equals 3 and other distance shenanigans

How does `util_math_dist_fast()` even work?

The distance between two points on a flat surface is the square root of the sum of coordinates' deltas squared and there's no square root in sight!
And even if it was known by then, most probably fast inverse square root trick wouldn't work on the machines MOO1 was targeting.

So here's how you do it.

```
int util_math_dist_fast(int x0, int y0, int x1, int y1)
{
    int dx = abs(x1 - x0);
    int dy = abs(y1 - y0);

    if (dx > dy)
        return (dx + dy / 2);
    else
        return (dy + dx / 2);
}
```

And it works.

The most extreme case is when dx and dy both equal 1, then the integer division makes 0 out of dx and you get 41.2% relative error.

But other than that it averages down to about 7-8% relative error.

In best circumstances (dx equals dy) it's about 5.5% and I couldn't make the error higher than 10.56%.

Yeah, I know, you need to go down to really cheap MCUs to resort to such tricks nowadays. If you declared pi to be 3 "to make calculations easier", you could still be sure it's a twice more accurate approach than this formula!

It's even more unacceptable than my clickbait header for this part!
But it is acceptable.

Of course now you can't unsee it, but think of the countless times the game was played since 1993.

Multiply that by 100 of turns (lowballing). Multiply that by roughly 6000 times it's called each turn (also lowballing for a huge map and 5 rivals).

This function may have run trillion times and nobody noticed.

It is acceptable.

If by chance you have pi declared in your code, set it to 3 and look if anything breaks from the player's standpoint.

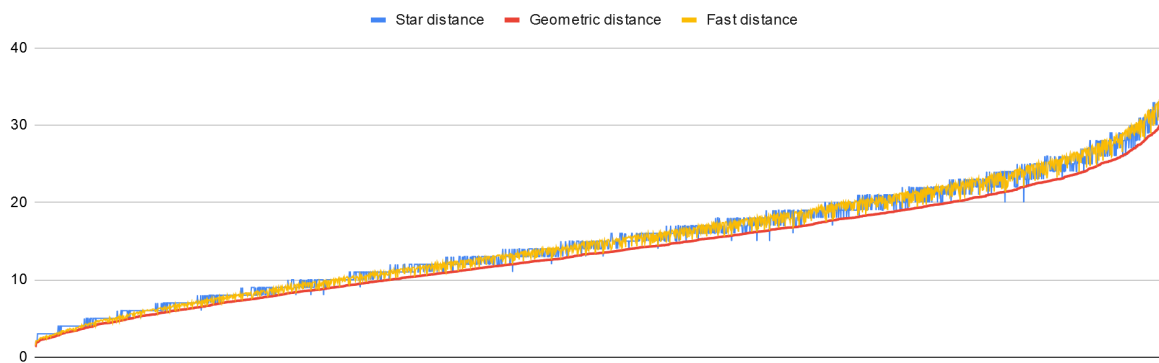
If you have to choose between speed and precision, try speed first.

There is one fuel-related oddity to keep in mind though.

There are two sets of distance units in-game, not fully compatible with each other.

In-code we'll see units of distance, the x and y coordinates of this or that, usually calculated by the aforementioned `util_math_dist_fast()`.

In-game all we see are distances in parsecs between the stars. They are calculated once and for all at the map generation (`init_star_dist()` in `game/game_aux.c`) using a different method, `util_math_dist_steps()`.



(the difference between star distance, fast distance/10.0 and geometric distance/10.0 on a huge map)

Fast distance is mainly used for visibility and many other things.
 Star distance is used for flight range and flight speed calculation.
 And they differ, differ a lot in some cases as can be seen from the graph.

As a rule of thumb 10 units roughly correspond to 1 parsec, but not always.
 You can easily have cases when the star distance is 6 parsecs, but the fast distance is 48 or when the star distance is 9, but the fast distance is 101.

dist_steps() isn't more accurate than *dist_fast()*, as you can see on the graph.
 I tried replacing *dist_steps()* with *dist_fast()* (dividing the result by 10 and adding 1 if the remainder is 5 or more) and the map looked as usual.

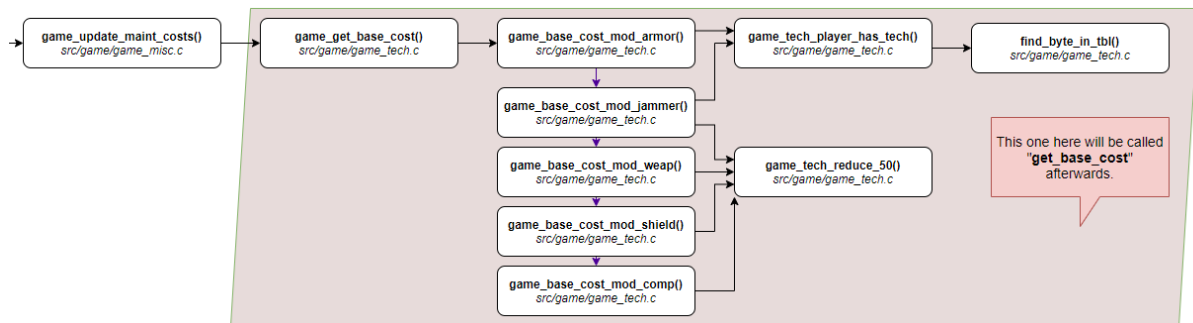
I'm stumped by this one.

Futile elegance of base maintenance costs

An empire checks how much it has to pay for what it already owns is by running *game_update_maint_costs()*.

For each player in the game the following is calculated:

- 1) the cost of all owned ships is summed up and then divided by 50, then the maintenance cost (100 BC) of each stargate the player owns is added — this is the ship maintenance in BC per turn;
 - a) technically, if it's larger than 32000 BC (*game_num_max_ship_maint*), then 32000 BC is used as the maintenance cost, but getting up there requires some very serious dedication;
- 2) the number of missile bases the player has on all planets is summed up, multiplied by the cost of the base and divided by 50 — this is bases' maintenance in BC per turn;
 - a) and AI gets to cheat here, on easy difficulty bases cost 0.9 of what they'd cost you, 0.8 on normal, 0.7 on hard and 0.5 on impossible;
 - b) if the final cost of the base turns out to be lower than 50 BC, it gets set to 50 BC.



But there's a whole bunch of other functions called from there, *get_tech_reduce_50()*, *get_base_cost_mod_shield()*, etc. What's the deal?

So the actual cost of the missile base is calculated as if it was a special large-sized ship without engines.

It's 60 percent of the sum of:

- large hull cost plus large hull armor cost, all of it multiplied by tech multiplier and divided by 1500
- weapons cost multiplied by tech multiplier and divided by 1000
- shield cost as if for a large ship multiplied by tech multiplier and divided by 1000 plus power requirements of a large ship shield, divided by 10
- computer cost as if for a large ship multiplied by tech multiplier and divided by 1000 plus power requirements of a large ship computer, divided by 10
- jammer cost as if for a large ship multiplied by tech multiplier and divided by 1000 plus power requirements of a large ship computer, divided by 10

The multiplier used here means reducing the cost by 50% each 10 levels you have in the tech.

But how do you do it without floating point arithmetics?

Witness the power of lookup tables:

```

const uint8_t tech_reduce_50percent_per_10pts[51] = {
    100, 93, 87, 81, 76, 71, 66, 62, 58, 54,
    50, 47, 44, 41, 38, 35, 33, 31, 29, 27,
    25, 23, 22, 20, 19, 18, 16, 15, 14, 13,
    13, 12, 11, 10, 9, 9, 8, 8, 7, 7,
    6, 6, 5, 5, 5, 4, 4, 4, 4, 3,
    3
};
  
```

Each ten points it's indeed roughly two times less. Each five points it's roughly 70% of what it was.

The only pesky thing left is that multiplying by it still increases the result, but that's what these "divided by 1000" are for.

No level in this tech field, you effectively divide by 10. Ten levels in this tech field, you effectively divide by 20.
Easy.

Now I scoured all contemporary user strategy guides for MOO1 that I could find and I didn't see anyone ever mentioning that bases become more expensive with the tech.

Nobody bothered to do this, despite the cost breakdown for the player being clearly displayed for everyone to see on the table 8-1 on page 133 of "Master of Orion Strategy Guide" by Emrich and Hughes and explained further on pages 134 and 135.

In MOO2 some players do optimize for the cost of the early scout, but in MOO1 seemingly nobody does that even for ships, and the design screen does display the cost there. And it's not like the player can even design the missile bases (may be OP, though, see Stars in Shadow).

I'm firmly of the opinion that this whole base maintenance count could've been replaced by something much more simple (let's say $(10 + \text{average\_tech\_level}) * 8$) and nobody would've found the difference.

What is not seen and not felt by the player may as well not exist at all.

Tech groups and ranking by the AI

This is the table of techs that's referenced in espionage and diplomatic sections.

Field	Level	Name	Group id	Intrinsic value	Notes
Computers	34	Hyperspace communications	0	0	Essentials. All of them are equally alluring for the AI.
Computers	46	Oracle interface	0	0	
Computers	49	Technology nullifier	0	0	
Force fields	16	Repulsor beam	0	0	
Force fields	27	Cloaking device	0	0	
Force fields	37	Stasis field	0	0	
Force fields	43	Black hole generator	0	0	
Planetology	16	Soil enrichment	0	0	
Planetology	17	Bio toxin antidote	0	0	
Planetology	21	Cloning	0	0	
Planetology	22	Atmospheric terraforming	0	0	
Planetology	30	Advanced soil enrichment	0	0	
Planetology	36	Universal antidote	0	0	
Planetology	42	Advanced cloning	0	0	
Propulsion	10	Inertial stabilizer	0	0	
Propulsion	20	Warp dissipator	0	0	

Propulsion	27	Intergalactic star gates	0	0	
Propulsion	34	High energy focus	0	0	
Propulsion	38	Sub-space teleporter	0	0	
Propulsion	43	Sub-space interdicator	0	0	
Propulsion	45	Combat transporters	0	0	
Propulsion	46	Inertial nullifier	0	0	
Propulsion	50	Displacement device	0	0	
Computers	2	Ecm jammer mark I	1	1	Jammers.
Computers	7	Ecm jammer mark II	1	2	
Computers	12	Ecm jammer mark III	1	3	
Computers	17	Ecm jammer mark IV	1	4	
Computers	22	Ecm jammer mark V	1	5	
Computers	27	Ecm jammer mark VI	1	6	
Computers	32	Ecm jammer mark VII	1	7	
Computers	37	Ecm jammer mark VIII	1	8	
Computers	42	Ecm jammer mark IX	1	9	
Computers	47	Ecm jammer mark X	1	10	
Computers	5	Battle computer mark II	2	1	Battle computers.
Computers	10	Battle computer mark III	2	2	
Computers	15	Battle computer mark IV	2	3	
Computers	20	Battle computer mark V	2	4	
Computers	25	Battle computer mark VI	2	5	
Computers	30	Battle computer mark VII	2	6	
Computers	35	Battle computer mark VIII	2	7	
Computers	40	Battle computer mark IX	2	8	
Computers	45	Battle computer mark X	2	9	
Computers	50	Battle computer mark XI	2	10	
Computers	8	Improved robotic controls III	3	1	More factories.
Computers	18	Improved robotic controls IV	3	2	
Computers	28	Improved robotic controls V	3	3	
Computers	38	Improved robotic controls VII	3	4	
Computers	48	Improved robotic controls VII	3	5	
Computers	4	Deep space scanner	4	1	Scanners.
Computers	13	Improved space scanner	4	2	
Computers	23	Advanced space scanner	4	3	
Construction	5	Reduced industrial waste 80%	5	1	Reduced pollution.

Construction	15	Reduced industrial waste 60%	5	2	Worthless for Silicoids.
Construction	25	Reduced industrial waste 40%	5	3	
Construction	35	Reduced industrial waste 20%	5	4	
Construction	45	Industrial waste elimination	5	5	
Construction	3	Improved industrial 9	6	1	Cheaper factories.
Construction	8	Improved industrial 8	6	2	
Construction	13	Improved industrial 7	6	3	
Construction	18	Improved industrial 6	6	4	
Construction	23	Improved industrial 5	6	5	
Construction	28	Improved industrial 4	6	6	
Construction	33	Improved industrial 3	6	7	
Construction	38	Improved industrial 2	6	8	
Construction	10	Duralloy armor	7	1	Ship armor.
Construction	17	Zortium armor	7	2	
Construction	26	Andrium armor	7	3	
Construction	34	Tritanium armor	7	4	
Construction	42	Adamantium armor	7	5	
Construction	50	Neutronium armor	7	6	
Construction	14	Automated repair system	8	1	Ship in-battle repair.
Construction	36	Advanced damage control	8	2	
Force fields	4	Class II deflector shields	9	1	Ship shields.
Force fields	10	Class III deflector shields	9	2	
Force fields	14	Class IV deflector shields	9	3	
Force fields	20	Class V deflector shields	9	4	
Force fields	24	Class VI deflector shields	9	5	
Force fields	30	Class VII deflector shields	9	6	
Force fields	34	Class IX deflector shields	9	7	
Force fields	40	Class XI deflector shields	9	8	
Force fields	44	Class XIII deflector shields	9	9	
Force fields	50	Class XV deflector shields	9	10	
Force fields	12	Class V planetary shield	10	1	Planetary shields.
Force fields	22	Class X planetary shield	10	2	
Force fields	32	Class XV planetary shield	10	3	
Force fields	42	Class XX planetary shield	10	4	

Weapons	6	Anti-missile rockets	11	1	Anti-missile specials.
Force fields	31	Zyro shield	11	2	
Force fields	46	Lightning shield	11	3	
Planetology	2	Improved terraforming +10	12	1	More population.
Planetology	8	Improved terraforming +20	12	2	
Planetology	14	Improved terraforming +30	12	3	
Planetology	20	Improved terraforming +40	12	4	
Planetology	26	Improved terraforming +50	12	5	
Planetology	32	Improved terraforming +60	12	6	
Planetology	38	Improved terraforming +80	12	7	
Planetology	44	Improved terraforming +100	12	8	
Planetology	50	Complete terraforming	12	9	Colonization tech. Ignored by Silicoids.
Planetology	3	Controlled barren environment	13	1	
Planetology	6	Controlled tundra environment	13	2	
Planetology	9	Controlled dead environment	13	3	
Planetology	12	Controlled inferno environment	13	4	
Planetology	15	Controlled toxic environment	13	5	
Planetology	18	Controlled radiated environment	13	6	Pollution removal tech. Ignored by Silicoids.
Planetology	5	Improved eco restoration	14	1	
Planetology	13	Enhanced eco restoration	14	1	
Planetology	24	Advanced eco restoration	14	2	
Planetology	34	Complete eco restoration	14	3	Troop armor.
Construction	11	Battle suits	15	1	
Construction	24	Armored exoskeleton	15	2	
Construction	40	Powered armor	15	3	Troop shields.
Force fields	8	Personal deflector shield	16	1	
Force fields	21	Personal absorption shield	16	2	
Force fields	38	Personal barrier shield	16	3	Ship engines.
Propulsion	6	Nuclear engines	17	1	
Propulsion	12	Sub-light drives	17	2	
Propulsion	18	Fusion drives	17	3	
Propulsion	24	Impulse drives	17	4	
Propulsion	30	Ion drives	17	5	
Propulsion	36	Anti-matter drives	17	6	

Propulsion	42	Inter-phased drives	17	7	
Propulsion	48	Hyper drives	17	8	
Propulsion	3	Hydrogen fuel cells	18	1	Fuel tanks.
Propulsion	5	Deuterium fuel cells	18	2	
Propulsion	9	Irridium fuel cells	18	3	
Propulsion	14	Dotomite crystals	18	4	
Propulsion	19	Uridium fuel cells	18	5	
Propulsion	23	Reajax II fuel cells	18	6	
Propulsion	29	Trilithium crystals	18	7	
Propulsion	41	Thorium cells	18	8	
Weapons	5	Gatling laser	19	1	Beam weapons.
Weapons	10	Ion cannon	19	2	
Weapons	15	Neutron blaster	19	3	
Weapons	17	Graviton beam	19	4	
Weapons	20	Fusion beam	19	5	
Weapons	25	Megabolt cannon	19	6	
Weapons	26	Phasor	19	7	
Weapons	30	Tachyon beam	19	8	
Weapons	28	Auto-blaster	19	9	
Weapons	35	Plasma cannon	19	10	
Weapons	37	Disruptor	19	11	
Weapons	38	Pulse phasor	19	12	
Weapons	45	Tri-focus plasma cannon	19	13	
Weapons	46	Stellar converter	19	14	
Weapons	48	Mauler device	19	15	
Weapons	36	Death ray	19	50	
Weapons	4	Hyper-V rockets	20	1	Missile weapons.
Weapons	8	Hyper-X rockets	20	2	
Weapons	11	Scatter pack V rockets	20	2	
Weapons	14	Merculite missiles	20	3	
Weapons	18	Stinger missiles	20	4	
Weapons	23	Anti-matter torpedoes	20	5	
Weapons	27	Scatter pack VII missiles	20	5	
Weapons	29	Pulson missiles	20	6	
Weapons	34	Hercular missiles	20	7	
Weapons	40	Hellfire torpedoes	20	8	
Weapons	41	Zeon missiles	20	9	

Weapons	43	Proton torpedoes	20	10	
Weapons	44	Scatter pack X missiles	20	10	
Weapons	50	Plasma torpedoes	20	11	
Weapons	2	Hand lasers	21	1	Troop weapons.
Weapons	12	Ion rifle	21	2	
Weapons	24	Fusion rifle	21	3	
Weapons	31	Hand phasor	21	4	
Weapons	42	Plasma rifle	21	5	
Weapons	9	Fusion bomb	22	1	Bombs.
Weapons	16	Anti-matter bomb	22	2	
Weapons	22	Omega-V bomb	22	3	
Weapons	39	Neutronium bomb	22	4	
Planetology	10	Death spores	23	1	Bio-bombs.
Planetology	27	Doom virus	23	2	
Planetology	40	Bio terminator	23	3	
Propulsion	16	Energy pulsar	24	1	Pulsar specials.
Propulsion	40	Ionic pulsar	24	2	
Weapons	21	Ion stream projector	25	1	Armor stripping specials.
Weapons	47	Neutron stream projector	25	2	
Weapons	7	Neutron pellet gun	30	2	Shield-breaking weapons.
Weapons	13	Mass driver	30	3	
Weapons	19	Hard beam	30	4	
Weapons	32	Gauss autocannon	30	6	
Weapons	33	Particle beam	30	7	

States of the AI

Well, for the purposes of interacting with the player an AI may be in one of 81 states. Just glancing over the tables will give you a good understanding of what AI is capable of.

I thought about putting it into the main writeup, but it broke the flow too much.

	Establishing contact
Xenophobic	14
Ruthless	15
Aggressive	16
Erratic	17
Honorable	18
Pacifistic	19

	Greeting
Thank you for sweetening the deal with BCs	1
Thank you for sweetening the deal with a tech	2
Thank you for attacking a common enemy	3
Warm greeting	22
Cold greeting	23
After voting against this AI in an undecided election	79
After voting for this AI in an undecided election	80

As you'll notice, refusals are largely +30 from the warnings, threats to break a treaty are +38 from the warnings, and the notifications about the broken treaty are +46 from the warnings.

	Warning	Refusal	Threat to break a treaty	Breaking treaty
For distrust	N/A	33	N/A	N/A
For espionage	4	34	42	50
For being framed for espionage	5	35	43	51
For sabotage	6	36	44	52
For being framed for sabotage	7	37	45	53
For amassing ships at the border	8	38	46	54
For attacking ships	9	39	47	55
For attacking a colony	10	40	48	56
For using bio weapons	11	41	49	57
For having too many colonies	12	N/A	N/A	N/A

The game tries to keep what was the original reason each turn, so all treaty breaks get reduced (by subtracting 46) to their original causes.

However, threats get 30 subtracted (as should have been done for refusals), leading to 12-19 range.

Most probably it's a bug as the original "blunder" as the game calls them becomes a greeting, a warning for having too many colonies (in case of espionage) or war declaration (in case of being framed for espionage).

Refusals do not get reduced to the original causes for some reason.

	War declaration
General	13
NAP or alliance broken in process	32

Final war for defying senate	59
For being hated	60
By an erratic empire for no reason	61

Diplomatic state 0 is default and means war too.

	Miscellaneous
General refusal	31
Ceasefire	70
Ceasefire with a money gift	71
Ceasefire with a tech gift	72
Response to breaking a treaty	73

State 72 and the messages associated with it never get used.

State 70 is used instead both for a simple ceasefire and a ceasefire with a tech as a sweetener.

	Audience end
Ambassador recalled due to war	20
Ambassador recalled due to annoyance	21
For being annoyed	74

	Treaty proposal	Agreement
For a NAP	24	62
For an alliance	25	63
For a trade treaty	26	64
For a peace treaty	30	65

	Bounty proposal	Bounty delivery
Bounty for attacking a 3rd party	28	58

In Master of Orion 1 bounties are never paid due to a bug.

	Request	Refusal
Tech trade	29	75

	Request	Treaty break
--	---------	--------------

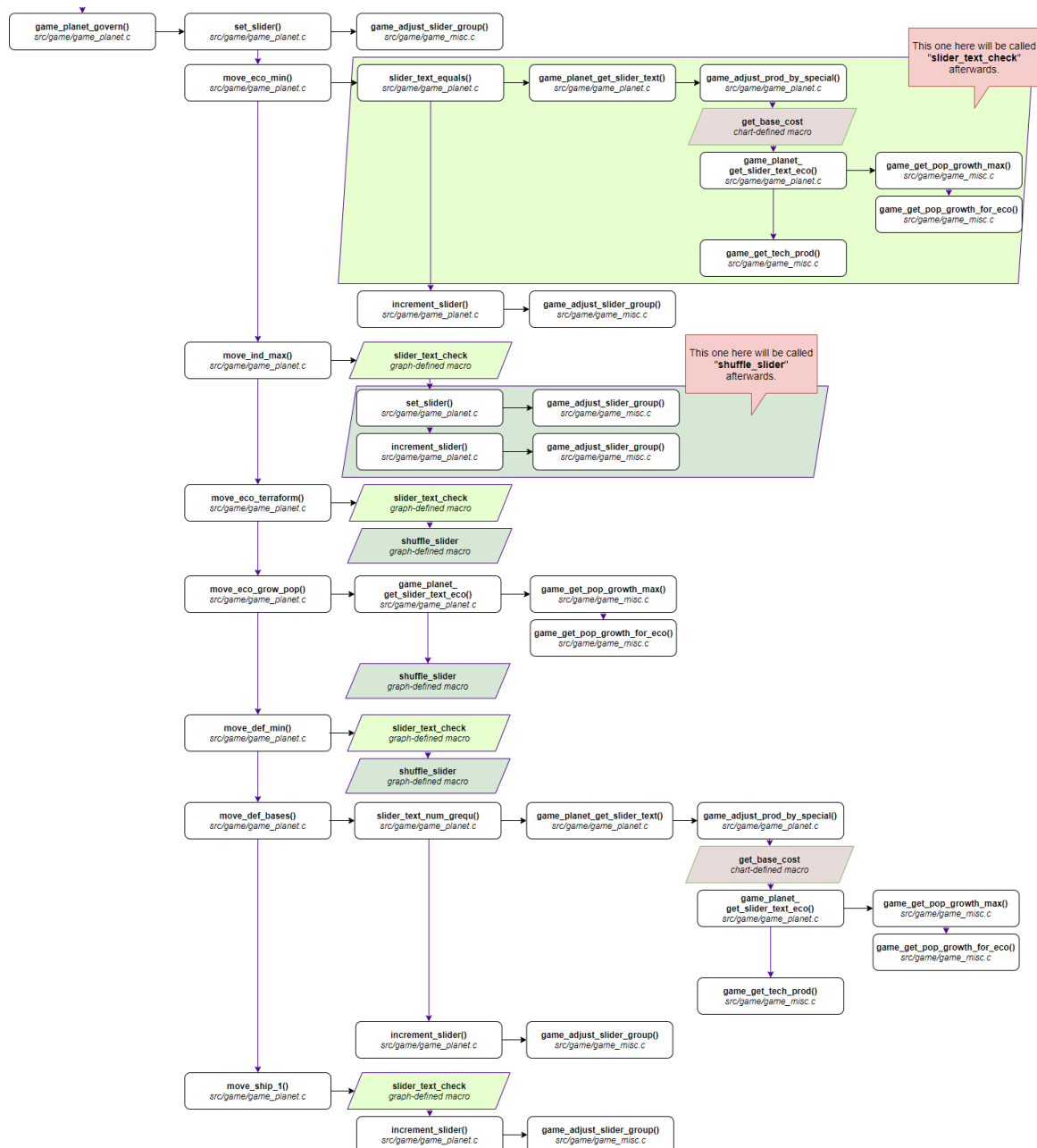
To honor the alliance and declare war on a 3rd party	76	N/A
For not declaring war when asked to in an alliance	N/A	77

	Request	Agreement	Refusal
To a BC or tech demand	N/A	66	69
To attack a 3rd party	78	67	N/A
To break treaty with a 3rd party	27	68	N/A

Planetary governor

Technically, it's an AI too. Just the AI that the player invokes.

Here's how it looks ([full res](#)):



The planetary governor is an apocryphal addition written by Kilgore Trout. Contrary to what the p3 phase looks like, it's much more human-like.

There are effectively four eco stances in the governor:

- grow population before building defences
- grow population before doing what was ordered previously
- keep eco spending at “clean” level
- do not touch eco spending set by the player, whatever it is

And there are three stances what to do with the excess production:

- spend it on research
- spend it on industry
- spend it on ships

It works as you'd expect it to work from the human standpoint.

The code locks or moves sliders like the human can do, "looks" at the values displayed and moves the slider back if necessary.

For example, *move_ind_max()* and similar functions work by comparing the slider text with "RESERVE" or "MAX" (technically, *game_str_sm_indres* and *game_str_sm_indmax*). Instead of being tied to the actual logic of the game, it's bolted on to the UI. Well, not actually, but close to that.

Kilgore Trout explains much better than me how it works:

```
/**
 * Govern the colony.
 * - First, remember the old ecology slider position.
 * - Then if mode is not "don't touch", set ecology to minimum that avoids waste.
 * - Then if mode is "don't decrease" and old value was larger, restore it.
 * - Then move industry slider until maximum or reserve achieved.
 * - Then if mode is not "do not touch", set ecology to terraform.
 * - Then if mode is "grow before defense", set ecology to grow population.
 * - Then set defense to upgrade bases or build shield.
 * - Then increase defense to build up to target amount of missile bases.
 * - Then if technology is present and governor is allowed to build stargates,
 *   set shipbuilding to minimum required to build one.
 * - Then if mode is "grow before last", set ecology to grow population.
 * - If all of the above are finished, do research, ship building or reserve.
 *
 * This works by moving slider by 1 unit (1%) until desired results happen. Implemented
 * this way to avoid duplication of planet production logic.
 */
```

So, it's cool and all that, but it's just a feature, right?

No. I don't think it's "just" a feature.

The p3 phase we've read about earlier, is essentially mindless reflexes.

This part that could, if expanded a bit more to cover for traits, actually replace colony spending routines, works with a discoverable intent.

Well worth contemplating a bit.

How to familiarize yourself with an existing codebase

That's not strictly related to Master of Orion 1, but it's a topic well worth covering.

Be it 100m or an open-source project you want to dabble in, or even a codebase you've been told to support at work, dealing with someone else's code is painful, yet oftentimes necessary.

Having dealt with that a bit, I'd like to offer my non-authoritative take on this problem, using my work on 100m as an example.

I may think of myself as a hotshot game designer, but as a programmer I'm deeply mediocre, so most likely you can repeat these steps with little to no challenge.

Understanding someone else's codebase is a grind, first and foremost.

You'll be discovering not only how the program does things, but also why and in what style. And as programmers tend to stick to their own style, later on things will get easier.

Set yourself up for the grind.

A lot of time it won't look like you made much progress, but believe me, you're getting through this.

Build

The grind starts with setting up the build environment as described in README or supplementary materials, no matter what you think about the toolset or OS or approach.

Make it compile, check if it runs, check if your experiences from running the program match the "proper" builds made by someone who ostensibly knows this stuff.

Once you get it working, make sure you can either easily replicate this build environment or that you can keep it pristine in a virtual machine, virtualenv or whatever.

You don't want to screw up your reference builds. Take your time.

Then you may be compelled to modify your build environment to better suit your habits, idiosyncrasies or testing requirements.

For example, 1oom is best built on Linux for Linux.

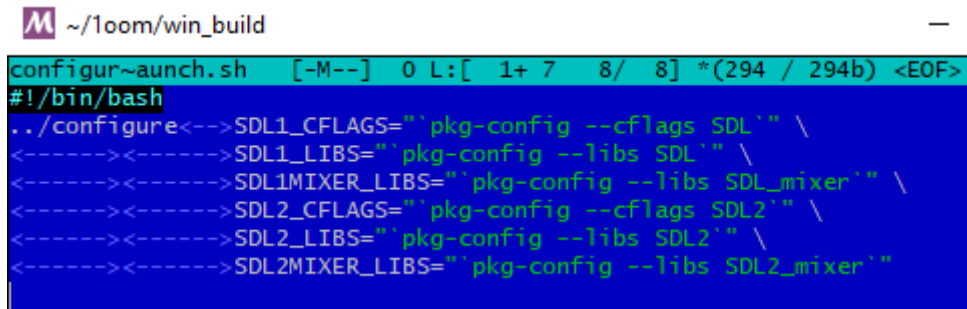
That's fine. I, for one, never left Linux. Linux left me.

Therefore, I used MSYS2 to build 1oom for Windows as I'm too mediocre to follow Kilgore Trout's recipes in the README for cross-compilation.

There is a subtle difference between "native" MSYS2 packages and mingw.

After a bit of troubleshooting, I ended up using *autoconf*, *automake-wrapper* and *make* from MSYS2, but *SDL*, *SDL_mixer*, *SDL2*, *SDL2_mixer*, *gcc* and *pkg-config* from mingw-w64-i686.

The only thing that was left was to point the configure script to proper libraries, and it's done using a small shell script:

A terminal window with a blue background and white text. The title bar shows a purple icon and the text '~/.loom/win_build'. The terminal content includes a shell prompt 'configur~aunch.sh [-M--] 0 L:[1+ 7 8/ 8] *(294 / 294b) <EOF>', followed by '#!/bin/bash' and several lines of configuration commands using 'pkg-config' to set SDL1 and SDL2 flags and libraries. The commands are: './configure<-->SDL1_CFLAGS=""pkg-config --cflags SDL' \', '<-----><----->SDL1_LIBS=""pkg-config --libs SDL' \', '<-----><----->SDL1MIXER_LIBS=""pkg-config --libs SDL_mixer' \', '<-----><----->SDL2_CFLAGS=""pkg-config --cflags SDL2' \', '<-----><----->SDL2_LIBS=""pkg-config --libs SDL2' \', and '<-----><----->SDL2MIXER_LIBS=""pkg-config --libs SDL2_mixer' '.

```
configur~aunch.sh [-M--] 0 L:[ 1+ 7 8/ 8] *(294 / 294b) <EOF>
#!/bin/bash
./configure<-->SDL1_CFLAGS=""pkg-config --cflags SDL' ' \
<-----><----->SDL1_LIBS=""pkg-config --libs SDL' ' \
<-----><----->SDL1MIXER_LIBS=""pkg-config --libs SDL_mixer' ' \
<-----><----->SDL2_CFLAGS=""pkg-config --cflags SDL2' ' \
<-----><----->SDL2_LIBS=""pkg-config --libs SDL2' ' \
<-----><----->SDL2MIXER_LIBS=""pkg-config --libs SDL2_mixer' '

```

And I could've never done that if I didn't understand how the game is built "as intended".

Only after everything is building and running in a way comfortable for you, only after that I can recommend to start messing with the code.

I do not recommend just "reading the code" unless you're looking for something trivial or are a very good programmer with an extensive domain and language expertise.

To check if you understood things correctly, you will often need to change the code and run it.

Hard to do it without a build environment.

Explore

Now you need an IDE.

It has to be able to do two things:

- search over all files in the project/directory
- be able to rename variables correctly in this language

I went with VS Code.

I don't like it and the last time I used it the renaming of variables in C was very wonky, but this time it was okay.

Use whatever you're familiar with.

I'd prefer MSVS, but it's non-trivial to make it work with pure C.

You may prefer your doom-emacs in evil mode, more power to you.

I'm not a real programmer anyway, so your choice will be at least as good as mine.

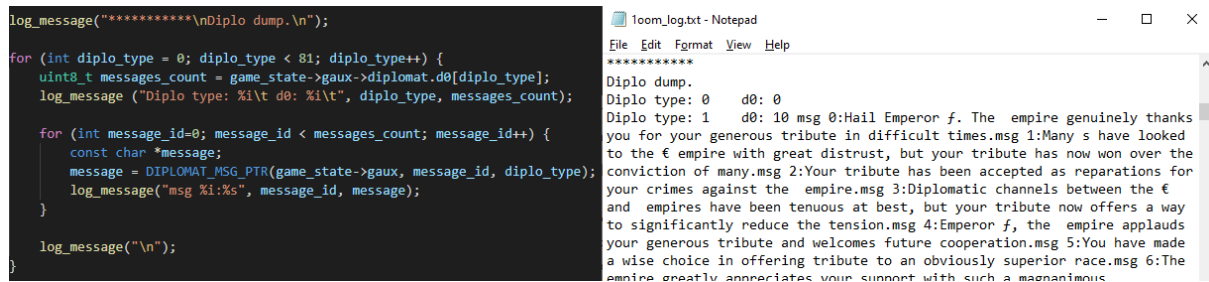
So, you're in, as hackers in corny movies say.

Your first task is to locate logging functions, methods, classes, calls, conventions, whatever. They usually have "log" or something like that in the name.

You will need to exfiltrate a lot of internal data from the program to find out what does this or that cryptic function do.

It's best to do it using the already established routes, they were made for a reason.

For example, I mapped all these 80 diplomatic states only by dumping the relevant messages as they are taken from original .LBX files and are not present in the code:



```
log_message("*****\nDiplo dump.\n");

for (int diplo_type = 0; diplo_type < 81; diplo_type++) {
    uint8_t messages_count = game_state->gaux->diplomats.d0[diplo_type];
    log_message ("Diplo type: %i\t d0: %i\t", diplo_type, messages_count);

    for (int message_id=0; message_id < messages_count; message_id++) {
        const char *message;
        message = DIPLOMAT_MSG_PTR(game_state->gaux, message_id, diplo_type);
        log_message("msg %i:%s", message_id, message);
    }

    log_message("\n");
}
```

100m_log.txt - Notepad

```
*****
Diplo dump.
Diplo type: 0      d0: 0
Diplo type: 1      d0: 10 msg 0:Hail Emperor f. The empire genuinely thanks
you for your generous tribute in difficult times.msg 1:Many s have looked
to the € empire with great distrust, but your tribute has now won over the
conviction of many.msg 2:Your tribute has been accepted as reparations for
your crimes against the empire.msg 3:Diplomatic channels between the €
and empires have been tenuous at best, but your tribute now offers a way
to significantly reduce the tension.msg 4:Emperor f, the empire applauds
your generous tribute and welcomes future cooperation.msg 5:You have made
a wise choice in offering tribute to an obviously superior race.msg 6:The
empire greatly appreciates your support with such a magnanimous
```

Now you need to find out where, as some youtubers say, “the real game begins”. As in, how exactly and through calling what functions the program gets to display what you expect to see on the screen.

There are two ways for this.

You can search for something that certainly should be in the game, like “missile”.

I did it right now and got quite a lot of results.

I like the first result in `game_ai_classic.c` file, it's `v = p->missile_bases ? 1 : 0;`

It doesn't matter what it means for now. What matters is that we're in `game_ai_classic_turn_p1_front_find_planet()` function.

It gets called from... another search... `game_ai_classic_turn_p1_front()`. Ok.

One more step “up”, we're in `game_ai_classic_turn_p1()`.

And this one gets called from... nothing?!

What about just “turn_p1(”, can we find that somewhere?

Ah, yes. `game_ai->turn_p1(g);` in `game_turn.c`.

Oh, I get it, different AIs can have different functions to be called as that `turn_p1` call.

Very crafty, Mr. Trout.

And it looks like we're where we want to be, there are various interesting calls around, `game_turn_diplo_adjust`, `game_turn_build_eco`, etc.

It's a very nice place called `game_turn_process()`. We'll have to note it.

And one last search, this time for `game_turn_process` and we end up in `main_do()` in `game.c`.

Now that's the entry point into the game and this is how it works!

Better write that somewhere. Or draw a map.

What I described is the manual unwinding of the call stack.

In more civilized environments it can be done automatically, just put a breakpoint where you want, run the program and look into the call stack when it stops.

Still, your circumstances and tooling available may vary.

Rename

Every programmer has an unique way of naming things.

Sometimes naming is enforced by project conventions, but usually be prepared to delve into the original coder's inner world.

That's why an IDE that can rename variables, structs, classes, etc is so valuable.

Consider this:

```
static bool game_audience_sub2(struct audience_s *au)
{
    struct game_s *g = au->g;
    player_id_t ph = au->ph, pa = au->pa, pf = PLAYER_NONE;
    empiretechorbit_t *eh = &(g->eto[ph]);
    empiretechorbit_t *ea = &(g->eto[pa]);
    for (player_id_t i = PLAYER_0; i < g->players; ++i) {
        if ((i != ph) && (i != pa) && (ea->treaty[i] >= TREATY_WAR) && (eh->treaty[i] == TREATY_ALLIANCE)) {
            pf = i;
        }
    }
}
```

It's the start of an audience subroutine and if you have a brain comparable to Kilgore Trout's, you got it all instantly.

But in this case I'm not sure why you're still reading this essay.

Mediocre individuals like me have to rely on ctrl+r,ctrl+r (in MSVS) or F2 (in VS Code) to make heads or tails out of this.

What's *empiretechorbit_t*? What does *game_s* hold?

It's like a sudoku puzzle, you work from the easiest parts and slowly, but surely gain understanding of what's what.

You got it wrong?

Don't worry, the next parts won't make sense, you'll scratch your head and find out where you mixed an AI player with a human player, like I just did (I'm writing this concurrently with describing the diplomacy phase).

And eventually the code from the screenshot above may turn into:

```

static bool game_audience_sub2(struct audience_s *audience)
{
    struct game_s *game_state = audience->g;

    player_id_t host_player = audience->ph;
    player_id_t guest_player = audience->pa;
    player_id_t framed_player = PLAYER_NONE;

    empire_state_t *host_empire = &(game_state->empire_state[host_player]);
    empire_state_t *attending_empire = &(game_state->empire_state[guest_player]);

    for (player_id_t player_id = PLAYER_0; player_id < game_state->players; ++player_id) {
        if ((player_id != host_player) && (player_id != guest_player) &&
            (attending_empire->treaty[player_id] >= TREATY_WAR) &&
            (host_empire->treaty[player_id] == TREATY_ALLIANCE)) {
            framed_player = player_id;
        }
    }
}

```

It's the same code.

I just renamed some variables to my liking and according to my knowledge.

Now it's clear what it's all about.

This code establishes three variables for player ids, two variables for the state of the guest and host empires.

Then it searches for some player who's at war or final war with the guest, but is allied with the host.

Further reading of the subroutine reveals that it's all about asking to break an alliance.

The code is now much more verbose though.

Probably you'd like to rename the variables to something else. So do it.

Needless to say that if you're not the only one working on the codebase, you should rename things only in your private exploratory branch.

However, try to recompile the project after each rename if feasible, at least when you're just starting.

An IDE may or may not be guaranteed to find all the references all the time in the language you're using. In some languages, it can not.

If you rename this, rename that and after that you get a weird compiler error, it's fairly hard to understand what went wrong and where.

Document

It's easy to feel powerless, to feel buried under the mountain of code.

There's still approximately nine million and forty three screens to cover, there are no damn comments and you start forgetting what you worked on yesterday, let alone last week!

That's normal.

Let's go over it step by step.

First of all, the code was written by the same humans as you and I.
If they could write it, you can understand it.

Just keep on grinding.
Note stupid bugs and grin, they did mistakes too, they are not demi-gods.
It is well within your reach, given enough time and dedication.
Note how stupid parts now make much more sense and grin at your former self, you're making progress.

Second, sure there are no damn comments.
Your anger is righteous. So do the right thing and comment the code yourself.
This code, your own code, whatever code you're working with.

Note your frustration when a comment explains trivial things, but is mum about what and why is done in this part.
Write comments that impart your understanding of the logic onto the reader.

Kilgore Trout's comments are very on the point and are never trivial. However, they are rare.
Write comments while you still remember what was hard to understand.

Third, document your understanding of the codebase.
Comments are good, but they are buried in the code.
Make a document that you yourself can read and reread and expand and fix and reread again.

Like this one, the one you were reading.
Were, because it just ended.

Have fun.